



*Personal Computer
Education Series*

Logo

by Logo Computer Systems, Inc.

IBM Program License Agreement

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THIS DISKETTE(S) OR CASSETTE(S) PACKAGE. OPENING THIS DISKETTE(S) OR CASSETTE(S) PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED; AND YOUR MONEY WILL BE REFUNDED.

IBM provides this program and licenses its use in the United States and Puerto Rico. You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

LICENSE

You may:

- a. use the program on a single machine;
- b. copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on the single machine (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked "copy protected.");
- c. modify the program and/or merge it into another program for your use on the single machine (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement.); and,
- d. transfer the program and license to another party if the other party agrees to accept the terms and conditions of this Agreement. If you transfer the program, you must at the same time either transfer all copies whether in printed or machine-readable form to the same party or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PROGRAM, OR ANY COPY, MODIFICATION OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION OR MERGED PORTION OF THE PROGRAM TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

TERM

The license is effective until terminated. You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

LIMITED WARRANTY

THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED PERSONAL COMPUTER DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Continued on inside back cover



*Personal Computer
Education Series*

Logo

by Logo Computer Systems, Inc.

First Edition (August 1983)

This product could include technical inaccuracies or typographical errors.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

The following paragraph applies only to the United States and Puerto Rico: International Business Machines Corporation provides this manual "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time and without notice.

Products are not stocked at the address below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM Personal Computer dealer.

A Reader's Comment Form is provided at the back of this publication. If this form has been removed, address comments to: IBM Corporation, Personal Computer, P.O. Box 1328-C, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

© Copyright International Business Machines Corporation 1983

© Copyright Logo Computer Systems, Inc. 1983

Preface

The purpose of the *IBM Personal Computer Logo Reference* is to give you a thorough understanding of the many features of Logo by expanding the information contained in *Logo: Programming with Turtle Graphics*, which comes with IBM Personal Computer Logo. The *Logo Reference* should be used to look things up and not as a guide for new users.

If your computer does not have the special equipment needed to try the turtle graphics commands, you should read through *Programming with Turtle Graphics* anyway to get an idea of how to construct and use Logo programs.

The *Logo Reference* can help you in several ways:

- Part 1 is an overview of Logo and assumes you have experimented with turtle graphics. Part 1 includes chapters on getting started with Logo, the Logo grammar, the Logo editor, Logo's file system, and how Logo deals with numbers.
- Part 1 also contains a chapter on developing a Logo project. This chapter will help you plan and write your own programs.
- Part 2 presents a detailed explanation of primitives listed alphabetically. You can refer directly to the alphabetic listing of primitives if you want to learn how a particular primitive works.

By consulting the *Reference Card*, which contains the complete Logo vocabulary classified into 12 categories, you will learn which primitive does what job.

- The appendixes cover several areas: information and error messages, procedures that can be useful tools, advanced technical information dealing with memory space, the list of ASCII codes, a description of the startup file feature, and information about DOS.
- A glossary defining major terms is provided.

The terms “disk”, “diskette”, and “fixed disk” are used throughout this manual. Where “diskette” is used, it applies only to diskette drives and diskettes. Where “fixed disk” is used, it applies only to the IBM nonremovable fixed disk drive. Where “disk” is used, it applies to both fixed disks and diskettes.

Green print is used throughout the *Logo Reference* to show what the computer types and what you type on the screen. Italicized words or terms indicate emphasis.

Contents

Part 1. General Information

Chapter 1. Getting Started	1-1
What You Need	1-3
Caring for Your Diskettes	1-4
Important Tips	1-5
IBM Personal Computer Keyboard	1-6
Starting Logo	1-11
Before You Begin	1-13
Copying the Logo Language Diskette	1-13
One-Drive System	1-15
Two-Drive System	1-17
Creating a File Diskette	1-19
One-Drive System	1-20
Two-Drive System	1-22
Using Logo with a Fixed Disk	1-24
 Chapter 2. Logo Grammar	 2-1
Procedures	2-3
Inputs to Procedures	2-5
Quotes, Colons, and Brackets	2-6
Commands and Operations	2-8
Variables	2-10
Global and Local Variables	2-12
Understanding a Logo Line	2-14
Words, Lists, and Special Characters	2-16

Chapter 3. Logo Editor	3-1
EDIT Command	3-3
How the Editor Works	3-4
Editing Actions	3-5
Cursor Motion	3-5
Inserting and Deleting	3-7
Scrolling the Screen	3-8
Exiting from the Editor	3-8
Outside the Editor	3-9
Using the Editor	3-9
 Chapter 4. File Handling	4-1
Naming Files	4-3
Disk File Names	4-3
Device Names	4-5
Permanent and Temporary Storage	4-6
Types of Files	4-6
Directory File	4-8
Program File	4-8
Organizing Your Procedures in	
Files	4-9
Screen File	4-11
Dribble File	4-12
Data File	4-13
Step 1: Creating a Data File	4-14
Step 2: Retrieving Information	4-17
Step 3: Changing Information	4-18
Binary File	4-19
Printing Files	4-20
Printing to the Printer	4-22

Chapter 5. How Logo Handles Numbers and Mathematical Operations	5-1
Numbers that Logo Recognizes	5-5
Floating-point Format Examples	5-5
Examples of Exponential Format of Numbers and the Equivalent Scientific Notation	5-8
How Logo Outputs Numbers	5-9
PRECISION and Logo Numbers	5-13
Order of Mathematical Operations	5-16
Special Prefix Operations	5-17
The — (Minus) Sign	5-18
Turtle Angles and Math Angles	5-19
Chapter 6. Developing a Logo Project	6-1
The Subgoals	6-3
Step 1: Story Generator	6-4
Step 2: An Arithmetic Problem	6-6
Step 3: Variation in Problems	6-8
Suggestions for Modifications	6-11
Program Listing	6-12

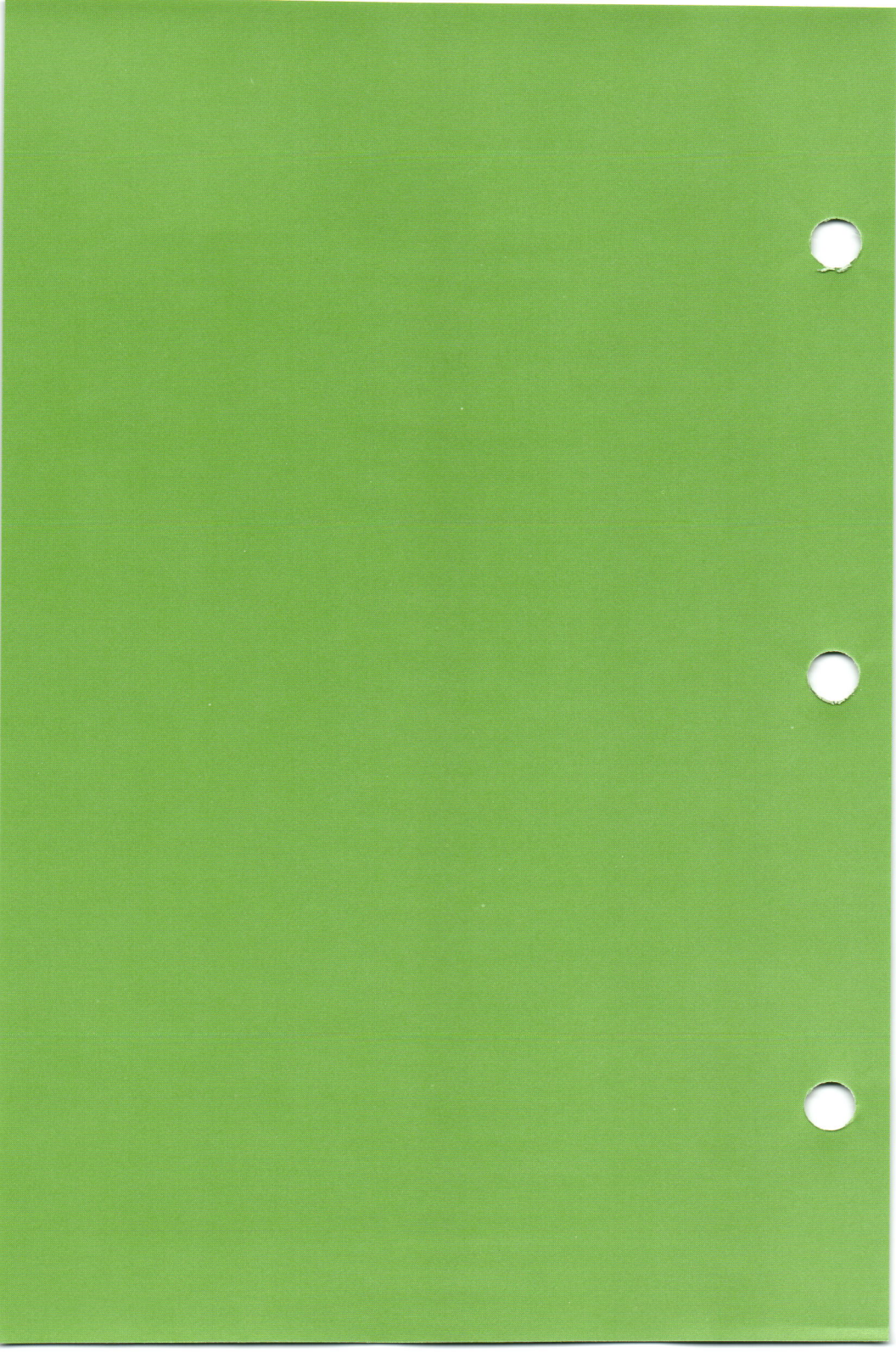
Part 2. Reference Information

Chapter 7. Logo Primitives	7-1
Introduction	7-3
Special Keys	7-9
Special Words	7-12
Infix Operations	7-13
.(Dot) Primitives	7-13
Alphabetical Listing of Primitives	7-13

Part 3. Appendixes

Appendix A. Messages	A-1
Appendix B. Useful Tools	B-1
Appendix C. How Logo Uses Memory	C-1
How Logo Allocates Memory	C-1
How Space is Used	C-3
Space-Saving Hints	C-4
Memory Allocation	C-5
Reserving High End	
Memory Space	C-5
Assembly Language Subroutines	C-8
Logo Memory Map	C-11
Appendix D. ASCII Character Codes	D-1
Extended Codes	D-5
Appendix E. Startup File Feature	E-1
Startup Variable	E-4
Appendix F. Your Logo Language Diskette and DOS	F-1
Replacing DOS on the Logo Diskette	F-1
How to Start Logo When DOS	
is Loaded	F-4
Editing the AUTOEXEC	F-4
Graphics Program	F-4
Mode Command	F-5
Glossary	Glossary-1
Index	Index-1

Part 1. General Information



Chapter 1. Getting Started

Contents

What You Need	1-3
Caring for Your Diskettes	1-4
Important Tips	1-5
IBM Personal Computer Keyboard	1-6
Starting Logo	1-11
Before You Begin	1-13
Copying the Logo Language Diskette	1-13
One-Drive System	1-15
Two-Drive System	1-17
Creating a File Diskette	1-19
One-Drive System	1-20
Two-Drive System	1-22
Using Logo with a Fixed Disk	1-24

What You Need

In order to use Logo, you need:

- An IBM Personal Computer XT or an IBM Personal Computer with a minimum of 128K bytes of random access memory.
- One diskette drive (minimum).
- A blank diskette.
- One or both of the following:
 - An IBM Color/Graphics Monitor Adapter with any color or black-and-white monitor or TV to do turtle graphics. An IBM Color Display or a direct drive monitor capable of displaying 40 or more columns is recommended to achieve quality turtle graphics.

Note: Composite color monitors or televisions may not produce true color images.

- An IBM Monochrome Display and Parallel Printer Adapter and monochrome monitor.

Note: You cannot use your IBM Monochrome Display for turtle graphics.

- The Logo Language Diskette, which is found at the back of the *Logo Reference* binder. A Logo Language Diskette contains the Logo program with samples and useful tools, the IBM Personal Computer Disk Operating System (DOS Version 2.00), the **FORMAT**, **DISKCOPY**, and **GRAPHICS** commands, and **AUTOEXEC.BAT**, a batch file which is automatically executed when you start the system.

These additions are optional:

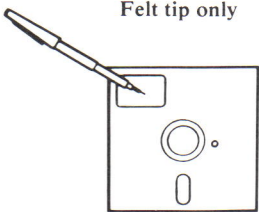
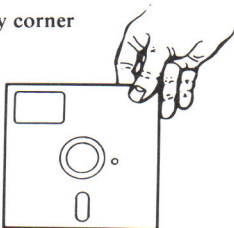
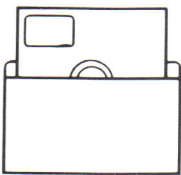
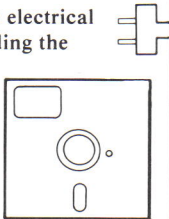
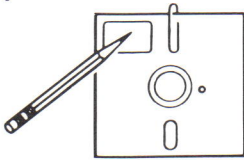
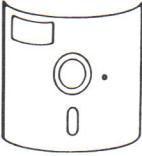
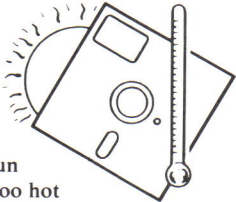
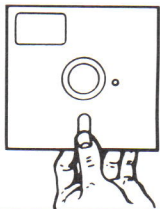
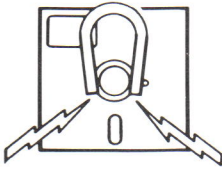
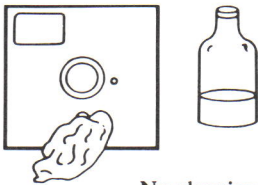
- A Parallel Printer Adapter.
- An IBM Graphics Printer or any printer compatible with your IBM Personal Computer.
- A Game Control Adapter to attach joysticks or paddles.
- An Asynchronous Communications Adapter. This can be used for a serial printer.

Caring for Your Diskettes

Diskettes are coated with magnetic material which is used to memorize programs and variables in your workspace. Parts of programs can be lost if this magnetic surface is damaged by heat, cold, or handling. To keep your diskette in good shape, observe these precautions:

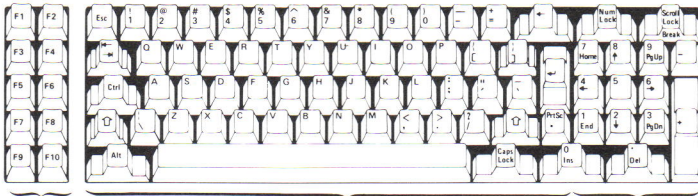
- Always place the diskettes back into the protective jackets after use.
- NEVER touch the magnetic surface of the diskettes.
- Keep your diskettes out of direct sunlight and away from other sources of heat. They may be stored at temperatures ranging from 40 to 125 degrees Fahrenheit (or 4 to 52 degrees Celsius).
- Use a felt tip pen when writing on diskette labels to avoid damaging the diskette inside the envelope.

Important Tips

Do's	
<p>Felt tip only</p> 	<p>Grasp by corner</p> 
<p>When not in use</p> 	<p>Keep it away from electrical equipment (including the telephone and the TV).</p> 
Don'ts	
<p>No pencils No clips No ballpoints</p> 	<p>Don't bend</p> 
<p>No sun Not too hot</p> 	<p>Don't touch diskette</p> 
<p>No magnets</p> 	<p>No cleaning</p> 

IBM Personal Computer Keyboard

Become familiar with your IBM Personal Computer keyboard. It has several kinds of keys.



**Function
keys**




Typewriter keyboard

**Numeric
keypad**






Character Keys



The character keys are letters of the alphabet, numbers, and punctuation marks.

Enter Key



Use the Enter  key when you want Logo to obey the instructions you've given it. The Enter  key tells Logo: "Now do what I just typed". Press the Enter  key after typing each line.



Shift Key

There is a Shift  key at each end of the keyboard. Holding down the Shift  key while pressing certain character keys changes these keys. For example, pressing ; (semicolon) with Shift  produces a : (colon). For the Shift  key to work, press the Shift  key first and keep it down while typing the character key.

Note: The  key is disabled once Logo starts up. All alphabet character keys appear in uppercase since Logo only understands its primitives in uppercase letters. Using the Shift  key with an alphabet character causes that letter to appear in lowercase. There is a primitive called SETCAPS which enables the Caps Lock function if you wish to use it.



Control Key

Unlike the Shift  key, a combination of the  key and certain character keys has an important effect that you cannot see.

For example, while holding down the  key, press the key labeled Break (also labeled ). The combination of the Ctrl-Break keys, called *Break*, tells Logo to stop what it is doing. Logo then prints

STOPPED!

? ■

Pressing the  key while holding down the  key interrupts whatever is running. Typing any key resumes running.

Another use of the **Ctrl** key is the **Ctrl** - **Alt** - **Del** key combination. This three-key combination performs a *system reset*, which is similar to switching the computer from off to on. If your Logo Language Diskette is in drive A at the time, the Logo program is loaded into memory.

Escape **Esc** Key

The **Esc** key is used to exit from the Logo Editor. This key is discussed along with other special editing keys in Chapter 3, "Logo Editor."

Bracket Keys

The left bracket **[** and right bracket **]** are used in Logo to group information in a list.


Parentheses Keys


The left parenthesis **(** and right parenthesis **)** keys are located on the 9 key and 0 key respectively, at the top of the keyboard. The **()** (parentheses) cannot be used in place of **[]** (brackets).


Cursor Keys

The cursor is the flashing box you see on the screen. On the numeric keypad, located on the right side of the keyboard, are keys with numbers 0 to 9.




The Cursor Up **8** key, on the 8 key, moves the cursor up to the previous line. This key can only be used in the Logo Editor.

The Cursor Down  key, on the 2 key, moves the cursor down to the next line. This key can only be used in the Logo Editor.



The Cursor Right  key, on the 6 key, moves the cursor one space to the right (forward).

The Cursor Left  key, on the 4 key, moves the cursor one space to the left (backward).


Num Lock Key

You can use the  key to set the numeric keypad so it works more like a calculator keypad. Pressing the  key shifts the numeric keypad into its own uppershift mode, so that you get the numbers 0 through 9 and the decimal point, as indicated on the keytops. Pressing  again will return the keypad to its normal cursor control mode.

Backspace Key

Do not confuse the Cursor Left  key with the Backspace , the large key on the top row of the keyboard. When you press this key, it moves the cursor backwards along a line and erases all characters passed over.

Delete Key

Pressing the  key once erases the character under the cursor.

Spacebar

The spacebar prints an invisible (but very important) character called a space. Logo uses spaces to separate words. For example, Logo reads THISISAWORD as a single word and reads THIS IS A WORD as four words.

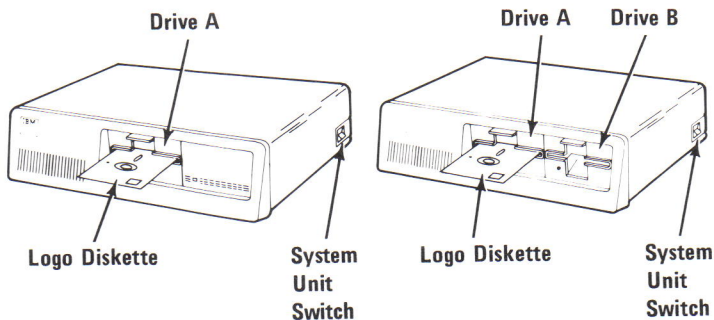
Pressing the spacebar inserts a space at the cursor's position. Thus, pressing the spacebar when the cursor is on the O of DOG produces D OG.

Function Keys

The keys labeled F1 to F10, located on the far left of the keyboard, are called *function keys*. Only the F1 to F5 keys are operative in Logo. See the Reference Card or "Special Keys" in Chapter 7 for their functions.

Starting Logo

1. Open the door of drive A (the left-hand drive) on your IBM Personal Computer.




2. Holding your Logo Language Diskette by the end with the label, insert it into drive A with the label facing up and close the drive door.
3. Find the System Unit Switch on your computer. (The System Unit Switch is located towards the back of the right side of the computer.) Switch it to On. Turn on your monitor or TV. (If you have an IBM Monochrome Display, it comes on with the computer.)

If your machine is already on, hold down the following three keys at the same time: **Ctrl** , **Alt** , and **Del** . As already noted, this three-key combination performs a system reset.

4. The drive A light will go on. After a few seconds, the light will go off and you will see a message on the screen asking you to enter today's date.


```
A>DATE
Current date is Tue 1-01-1980
Enter new date:
```


If today were June 20, 1983, you would type 6-20-83 and press the Enter  key. Enter the date now.

The drive light will go on and you will then see a message asking you to enter the time.

```
A>TIME  
Current time is 0:00:10.31  
Enter new time:
```

Use the 24-hour clock to record time.

If the time were 4:30 p.m., you would type 16:30 and press the Enter  key. Enter the time now.

The drive light will go on and you will see

```
A>GRAPHICS
```

This means that the *graphics* command is loaded into memory. It supports the IBM Graphics Printer.

```
A>LOGO
```

This command loads the Logo program into memory.

5. After a few seconds, the drive light will go off and the copyright statement and serial number will appear on the screen, as well as a message saying

```
WELCOME TO LOGO  
? ■
```

Below the message you will see a ? (question mark), the prompt symbol, followed by a flashing box ■.

When ? (question mark) is on the screen, Logo waits for you to type something. Whenever you type in response to the prompt symbol, you are at *top level*.

The flashing box ■ is the *cursor*. It appears when Logo wants you to type something and shows where the next character you type will appear.

Before you Begin

When you use your Logo Language Diskette for the first time, you must follow these procedures:

1. Copying the Logo Language Diskette.
2. Creating a file diskette.

These have to be done only when you start using Logo. After your first use of Logo, skip this section.


Copying the Logo Language Diskette

You should make a backup copy, store the master in a safe place, and always use the backup copy. If you damage or lose your backup copy, you can use your master to make a new backup copy.

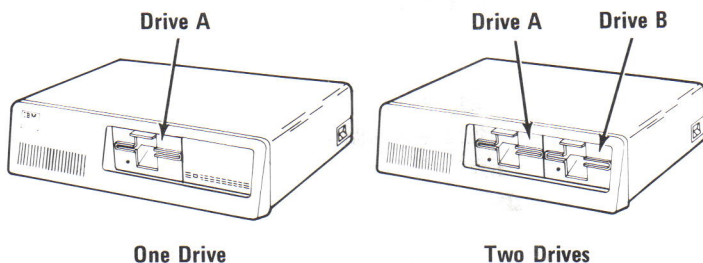
To make a copy, you need a blank diskette. Make sure your Logo Language Diskette has a write protect tab covering the notch on the diskette. This is important because if the diskettes get mixed up, this message is displayed:

Target diskette write protected
Correct, then strike any key

If you get this message, do the following:

1. Remove the Logo Language Diskette from the drive.
2. Insert the backup diskette into the drive.
3. Press any key. (You do *not* have to press the Enter  key.)



Before you begin to copy your Logo Language Diskette, check whether you have one or two drives. (See the illustration below.)



Note: If you have an IBM Personal Computer XT, follow the steps for a one-drive system.

One-Drive System

If you have one drive, do the following:

1. Start Logo as described in this chapter. Then press the Enter  key without removing the Logo Language Diskette. If Logo is already started, make sure your Logo Language Diskette is in the drive.
2. Type `.DOS` and press the Enter  key.
3. You will see the following DOS prompt on your screen:

`A>`

4. Type:

`DISKCOPY`

and press the Enter  key.

5. This message appears:

`Insert source diskette in drive A:`

`Strike any key when ready`

Because the Logo Language Diskette (the *source diskette*) is already in drive A:, you do *not* need to exchange diskettes.

6. Press any key. You will see this message:

`Copying 9 sectors per track, 1 side(s)`

7. The “in use” light will come on while the original diskette is being read and this message will be displayed:

Insert target diskette in drive A:

Strike any key when ready

Before touching a key:

- a. Remove the Logo Language Diskette.
- b. Insert the backup diskette (the blank diskette). This is the *target diskette*.
- c. *Now* press a key to tell DOS that the correct diskette has been inserted. You will see the message:

Formatting while copying

8. The “in use” light will come on while the backup diskette is being written. This message will be displayed:

Insert source diskette in drive A:

Strike any key when ready

Do this:

- a. Remove the backup diskette.
- b. Insert the Logo Language Diskette.
- c. Press any key only after the Logo Language Diskette is in the drive.

9. Repeat steps 7 and 8 until this message appears:

Copy complete

Copy another (Y/N)?

10. Type:

N

You *don't* have to press the Enter  key.



The DOS prompt, **A>**, is displayed.

11. Remove the diskette from the drive. With a felt tip pen, label and date your Logo Language Diskette Backup. Place a write-protect tab over the notch of the backup diskette, and store the master in a safe place. Always use your backup diskette. If you happen to lose or damage your backup diskette, you can retrieve your master and make a new backup.

Skip the next section and go to “Creating a File Diskette”.

Two-Drive System

If you have two drives, do the following:

1. Start Logo as described in this chapter. Press the Enter  key without removing the Logo Language Diskette. If Logo is already started, make sure your Logo Language Diskette is in the drive A.
2. Type **.DOS** and press the Enter  key.

3. You will see the following DOS prompt on your screen:

A>

4. Insert the blank diskette (the backup) into drive B.
5. Type:

DISKCOPY A: B:

and press the Enter  key.

6. You will see this message:

Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready

7. Press any key, because the backup diskette will already be in drive B. You will see this message:

Copying 9 sectors per track, 1 side(s)

Formatting while copying

Now the entire diskette is being copied from the diskette in drive A to the diskette in drive B. First one “in use” light will go on and then the other.

8. When the copy is made, you will see this message:

Copy complete

Copy another (Y/N)?

9. Type:

N

You *don't* have to press the Enter  key.

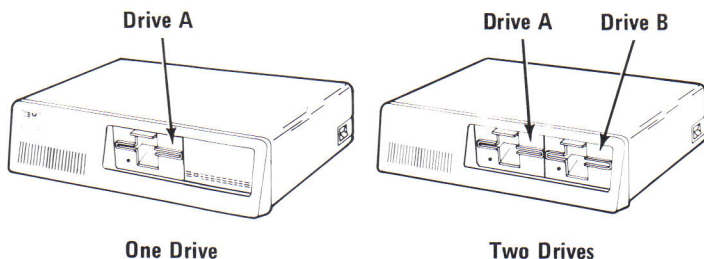
The DOS prompt, **A>**, is displayed.

10. Remove the backup diskette from drive B. With a felt tip pen, label and date your Logo Language Diskette Backup. Place a write-protect tab over the notch of the backup diskette and store the master in a safe place.

Creating a File Diskette

To save the programs that you write, you need a formatted diskette. Any blank diskette compatible with your IBM Personal Computer can be used as a Logo file diskette. Your Logo Language diskette contains the IBM Personal Computer Disk Operating System (DOS) Version 2.00 and the **FORMAT** Command. Only the **FORMAT** command from DOS 2.00 can be used to format a diskette compatible with the Logo language.



Before beginning to format a diskette, check whether you have one drive or two drives (See illustration below).




Note: If you have an IBM Personal Computer XT, follow the steps for a one-drive system.

One-Drive System

If you have one drive, follow these instructions to format a diskette. If you already have the DOS prompt on your screen, skip Steps 1 and 2:

1. Start Logo as described in this chapter. Without removing the Logo Language Diskette, press the Enter  key. If Logo is already started, make sure your Logo Language Diskette is in the drive A.
2. Type `.DOS` and press the Enter  key.
3. You will see the following DOS prompt on your screen:

`A>`

4. Type `FORMAT A:` and press the Enter  key.
5. You will see this message:

`Insert new diskette for drive A:
and strike any key when ready`

6. When the drive “in use” light is off, remove your Logo Language Diskette. Holding your blank diskette so that the notch is on the left side near the front, insert it into the drive and close the door.

7. When your blank diskette is inserted in the drive and the door is closed, press any key. You will see the drive light on and the message

Formatting...

will appear on the screen. In a few moments, you will see the message:


Formatting...Format complete

nnnnnnn bytes total disk space
nnnnnnn bytes available on disk

Format another (Y/N)?

If you have more blank diskettes to format, type **Y** and repeat steps 6 and 7 again.



8. When finished formatting diskettes, type **N**.
9. You will see the DOS prompt **A>** again on your screen. Remove the file diskette from the drive and label and date it with a felt tip pen.

You can now restart Logo by inserting your Logo Language Backup diskette into drive A, typing **LOGO**, and pressing Enter  key.


A>LOGO

Two-Drive System

If you have two drives, follow these instructions to format a diskette. If you already have the DOS prompt on your screen, skip Steps 1 and 2:

1. Start Logo as described in this chapter. Without removing the Logo Language Diskette, press the Enter  key. If Logo is already started, make sure your Logo Language Diskette is in the drive A.
2. Type `.DOS` and press the Enter  key.
3. You will see the following DOS prompt on your screen:

`A>`

4. Type `FORMAT B:` and press the Enter  key.
5. You will see this message:

`Insert new diskette for drive B:
and strike any key when ready`

6. Holding your blank diskette so the notch is on the left side near the front, insert it into drive B and close the drive door.

7. When your blank diskette is inserted into drive B and the door is closed, press any key. You then will see the light on drive B, and the message

Formatting...

will appear on the screen. In a few moments, you will see the message:


Formatting...Format complete

nnnnnnn bytes total disk space

nnnnnnn bytes available on disk

Format another (Y/N)?

If you have more blank diskettes to format, type **Y** and repeat Steps 6 and 7 again.

8. When finished formatting diskettes, type **N**.
9. You will see the DOS prompt **A>** again on your screen. Remove the file diskette from drive B and label and date it with a felt tip pen. Restart Logo by inserting your Logo Language Backup diskette into drive A, typing **LOGO** and pressing Enter .



A>LOGO

Using Logo with a Fixed Disk

If you wish to use Logo on an IBM Personal Computer XT or on an IBM Personal Computer with an Expansion Unit, you may do so in two ways:

1. Start Logo from your Logo Language Diskette in the diskette drive.
2. Copy Logo to the fixed disk and start it from there.

If you choose the second option, the fixed disk must have been configured and partitioned for use. The partition you use for Logo must be selected as the active one. See your *IBM Personal Computer DOS 2.00 Reference* for instructions on preparing the fixed disk. To transfer Logo to the fixed disk, follow these instructions:

1. Start Logo from your Logo Language Diskette.
2. Type `.DOS` and press the Enter  key.
3. In response to the DOS prompt `A>`, type `COPY LOGO.COM C:` and press the Enter  key.

`A>COPY LOGO.COM C:`

As long as Logo is not erased from the fixed disk, it may be started from there without using the Logo Language Diskette by typing `LOGO` in response to the DOS prompt

`C>LOGO`

Chapter 2. Logo Grammar

Contents

Procedures	2-3
Inputs to Procedures	2-5
Quotes, Colons, and Brackets	2-6
Commands and Operations	2-8
Variables	2-10
Global and Local Variables	2-12
Understanding a Logo Line	2-14
Words, Lists, and Special Characters	2-16

The Logo language is made up of building blocks. Certain rules must be followed as you put these blocks together. These rules are the “grammar” of the language. If Logo is to understand what you want it to do, you must give it proper instructions. This chapter presents some guidelines for you to follow as you prepare instructions for Logo.

Procedures

The building blocks of Logo are procedures and inputs to procedures. There are some procedures that Logo always knows because they are built into the system. These are called *primitives*. There is a complete list of them in Chapter 7, “Logo Primitives.”

For example, if you type

```
CLEARTEXT
```

the screen will be cleared of text. You haven’t defined CLEARTEXT but Logo already knows what to do.

There are some procedures that you define for yourself by using the TO or EDIT commands. There are many examples in both this book and in *Programming with Turtle Graphics*.

Here is a procedure definition.

```
TO WELCOME  
PRINT "HI  
END
```

The first and last lines have their own rules. The first line is called the *title line*. It must always begin with TO followed by the name of a procedure. The last line must always be the word END.

It is important to distinguish between defining a procedure and asking Logo to execute it. When we ask Logo to run a procedure, we say that we have made a *procedure call*.

For example, WELCOME contains a request to run a procedure PRINT (which happens to be a primitive).

There is another way of asking Logo to make a procedure call. The name is typed in when Logo is at top level (indicated by the question mark prompt at the left of the screen). We have already seen an example with CLEARTEXT. Here is another example.

```
?WELCOME  
HI
```

If you type in a word and Logo cannot find its definition, you get an error message. Suppose, for example, you haven't defined a procedure called KIYOKO.

```
?KIYOKO  
I DON'T KNOW HOW TO KIYOKO
```

Within a procedure definition, you can call a procedure you have previously defined.

```
TO WELCOME.FOREVER  
WELCOME  
WELCOME.FOREVER  
END
```

```
?WELCOME.FOREVER  
HI  
HI  
HI  
....
```

We say that WELCOME is a *subprocedure* of WELCOME.FOREVER. WELCOME.FOREVER is a *superprocedure* of WELCOME.

Inputs to Procedures

Some procedures need inputs. For example,

```
?PRINT "HI  
HI
```

Here is what happens if you don't include the input.

```
?PRINT  
NOT ENOUGH INPUTS TO PRINT
```

The input to **PRINT** can be a sentence instead of a word. The sentence must have brackets around it.

```
?PRINT [HAVE A NICE DAY]  
HAVE A NICE DAY
```

The procedures that you define can also have inputs. These variables have values you don't need to specify until the procedure is executed. Their names must be written on the title line after the name of the procedure. Each name must have a colon in front of it. For example,

```
TO BIGWELCOME :N  
PR "HI  
PR :N  
PR [HAVE A NICE DAY]  
END
```

The title line tells Logo that the procedure **BIGWELCOME** has a single input whose name is **N**. The body of the procedure contains three calls of the procedure **PRINT** (**PR** is the short form of **PRINT**). The second call uses the input **N**. Here is an example of a request to execute **BIGWELCOME** at top level.

```
?BIGWELCOME "SHARNEE  
HI  
SHARNEE  
HAVE A NICE DAY
```

Here, the input to BIGWELCOME is SHARNEE. Logo takes this to be the value of N when it executes the procedure.

Thus, PRINT :N becomes

PRINT "SHARNEE

Quotes, Colons, and Brackets

When you ask Logo to execute a procedure, you must be very careful when you write the inputs. Logo understands every word as a request to run a procedure unless you indicate otherwise. For example,

```
?BIGWELCOME SHARNEE  
I DON'T KNOW HOW TO SHARNEE
```

Logo thinks that SHARNEE is a procedure. Logo doesn't know how to execute SHARNEE because SHARNEE has no definition.

In this example, Logo is able to find the definition.

```
?BIGWELCOME SUM 31 28  
HI  
59  
HAVE A NICE DAY
```

SUM is a procedure that adds its inputs. Because it is a primitive, Logo can find its definition within the Logo system. Only Logo can do this because primitive definitions are written in a language that people cannot understand. Procedures used as inputs will be explained further in the next section.

In order to tell Logo that an input is not a request to run a procedure, you need to use certain characters in a special way.

A word preceded by a quotation mark (for example, "SHARNEE) tells Logo that the input is the word itself. We call this a *literal word*. Note that numbers don't have to be preceded by quotes.

A word preceded by a colon (for example, :N) tells Logo that the word is a variable and that the input will be the value of the variable.

A sequence of words surrounded by brackets (for example, [HAVE A NICE DAY]) indicates that the input is a *list*. Literal words inside a list don't usually have to be preceded by quotation marks. There is, however, a special kind of list called an *instruction list*, in which quotation marks are used. See the IF, REPEAT, and RUN commands in Chapter 7.

The use of these three special characters is illustrated earlier in this chapter with the definition of BIGWELCOME.

PRINT "HI tells Logo to display the word HI.

PRINT :N tells Logo to display the value of N when the procedure is executed.

PRINT [HAVE A NICE DAY] tells Logo to display the list HAVE A NICE DAY. Note that, while PRINT leaves out the brackets in its display, they have not really been removed. See the SHOW command in Chapter 7. On the other hand, the colon and quote are more like procedures with a word as input. See the THING operation in Chapter 7.

Commands and Operations

A procedure that attempts to communicate with another procedure is said to *output* a value when it is executed. This value must be received by another procedure, otherwise, Logo issues an error message. For example:

```
?SUM 31 28  
I DON'T KNOW WHAT TO DO WITH 59
```

There are two kinds of procedures in Logo: *operations* and *commands*. Operations output a value, and commands do not. This distinction is so important that, in Chapter 7, we indicate whether each primitive is a command or an operation.

For example, PRINT and MAKE are commands. RANDOM, SUM, and + are operations.

One of the main reasons for this distinction is that an operation can be written only as an input to a procedure. This means that in every Logo line the first word must be a command.

We have already seen an example with PRINT SUM 31 28. Here are some more examples:

```
PRINT RANDOM 2
```

RANDOM 2 is the input to PRINT. The input to RANDOM is 2. When RANDOM 2 is executed, the result is communicated to PRINT.

```
PRINT SUM 3 2  
5
```

The result of computing the procedure **SUM** with inputs 3 and 2 is communicated to **PRINT**.

```
PRINT SUM 3 PRODUCT 5 2
13
```

The output of **PRODUCT** is the second input to **SUM**.

Note that an operation can be an input to itself. For example:

```
?PRINT SUM 5 SUM 3 10
18
```

Here, **PRINT** has as its input the procedure **SUM** with 2 inputs: the number 5 and the procedure **SUM** with inputs 3 and 10.

If you try to use a command as an input, this is what happens:

```
PRINT MAKE "X 25
MAKE DIDN'T OUTPUT TO PRINT
```

You get the error message because **MAKE** is a command.

Up to now, we have considered only Logo primitives. However, all of the procedures you define yourself are also either commands or operations. For example, the procedure **BIGWELCOME** (defined previously) is a command. The **FLIP** procedure is an operation.

```
TO FLIP
IF RANDOM 2 = 0 [OUTPUT "H] [OUTPUT "T]
END
```

FLIP outputs the letter H if RANDOM 2 is 0 and the letter T if RANDOM 2 is 1. As with primitives, just typing the procedure name yields an error message.

```
?FLIP  
I DON'T KNOW WHAT TO DO WITH H
```

or

```
I DON'T KNOW WHAT TO DO WITH T
```

But we can have

```
?PRINT FLIP  
H
```

or

```
?PRINT FLIP  
T
```

Almost all of the procedures in *Programming with Turtle Graphics* are commands. However, procedures involving words, lists, and numbers are frequently operations. To construct your own operations, you will always use the OUTPUT command. For more information see the OUTPUT command in Chapter 7.

Variables

The best way to understand variables in Logo is to view them as containers with names on the outside and contents inside. The colon in front of a word tells Logo to make its contents available to the procedure. If you type

```
PRINT :JOHN
```

Logo looks for a container named JOHN. If it finds one, it looks inside the container and makes whatever it finds available to PRINT. PRINT then displays the contents (value) of JOHN on the screen. If it finds nothing, Logo prints the error message:

```
JOHN HAS NO VALUE
```

There are two ways of putting things or placing values inside these containers. The first, which we have already discussed, is by using procedures with inputs. The second is by using the MAKE command.

```
?MAKE "JOHN 25  
?PRINT :JOHN  
25
```

MAKE is a procedure needing two inputs: a literal word and a value which can be a word, a list, or a number. Here, it creates a container called JOHN and places 25 inside it. Note that MAKE does not display anything on the screen. It is PRINT that displays this value.

This is a good illustration of the difference between a quote and a colon. The first input to MAKE is "JOHN because it makes a variable whose name is JOHN. The input to PRINT is :JOHN because we want to display the value of JOHN.

Here are some more examples:

```
?MAKE "X "JOHN  
?PRINT :X  
JOHN  
?PRINT :JOHN  
25
```

MAKE now has two quoted words as inputs. It puts the literal word JOHN inside the container X. The contents of JOHN from the MAKE of the previous example are left undisturbed.

```
?MAKE "X [JOHN MARY]
?PRINT :X
JOHN MARY
```

We have created a variable with a list inside it. Here is another example:

```
?MAKE "N RANDOM 2
?PRINT :N
```

You will see either a 0 or 1 on the screen.

The second input of MAKE is now the output of a procedure, RANDOM 2, which generates 0 or 1 in a random way. Inside N is the number that results when Logo executes RANDOM 2.

Global and Local Variables

When Logo is at top level and you create a variable with MAKE, the variable remains in your workspace until you erase it. For this reason it is called a *global variable*. A variable that remains in the workspace only as long as a procedure is being executed is called a *local variable*. Variables that are defined as inputs to procedures are always local variables.

To see the difference change BIGWELCOME so it prints the date.

```
TO BIGWELCOME :N
PR :DATE
PR "HI
PR :N
PR [HAVE A NICE DAY]
END
```


Here DATE is a global variable that we haven't defined yet. If you try to run BIGWELCOME, you will get the error message

```
DATE HAS NO VALUE IN BIGWELCOME:
PR :DATE
```

We can use MAKE at top level to give DATE a value.

```
?MAKE "DATE [MARCH 23 1983]
?BIGWELCOME "SHARNEE
MARCH 23 1983
HI
SHARNEE
HAVE A NICE DAY
```

On the other hand, N is a local variable because it is an input to a procedure. It only contains the word SHARNEE while the procedure BIGWELCOME is being executed.

It is easy to forget that you have created a global variable. You can always check which ones are in your workspace by using the PONS (Print Out NameS) command. You can erase them by using the ERNS (ERase NameS) command. The following example shows that N is truly local and DATE is truly global.

```
?ERNS
?MAKE "DATE [MARCH 23 1983]
?BIGWELCOME "SHARNEE
...
?PONS
DATE IS [MARCH 23 1983]
```

A value is not displayed for N because N disappeared after BIGWELCOME stopped executing.



When you use **MAKE** inside a procedure definition the variable can be either local or global. If it is an input to the procedure, then it is local. If it is not an input, then it is global. You can always make a variable local to a procedure by using the **LOCAL** command. See Chapter 7, “Logo Primitives.”

Note that a procedure does not stop running when a subprocedure is called. Therefore, a variable that is local to a procedure can be used by its subprocedures.

Understanding a Logo Line

Procedure definitions consist of lines of instructions. These are called *Logo lines* because they can be much longer than the lines you see on your screen. For example:

```
?MAKE "MANYNAMES [BILL GARY JOHN JOE FR→  
ANK JUDY DAN JIM ERIC LOUISE]
```

The → (arrow) indicates that the next screen line is a continuation of the first Logo line. You get long lines like this by continuing to type without pressing the Enter  key. The arrow is automatically inserted, and the display continues on the next line. The Logo line ends as soon as you press Enter .

Logo lines can be complex. It is a good idea to understand them in the same way as we understand an English sentence. There, we use our knowledge of nouns and verbs to split the sentence up into meaningful pieces. Sometimes, we go a step further and split the pieces into smaller pieces. In the same way, we use our knowledge of the difference between commands and operations to split a Logo line into meaningful pieces.

Here are some guidelines to help you split Logo lines:

- Whenever you see a procedure name, be sure you know how many inputs it has and whether it is a command or an operation.
- The first word of a Logo line is *always* a command.
- An operation is *always* the input to another procedure.
- Be sure to account for every input to a procedure.
- When the inputs to a command have been accounted for, the next procedure must be another command.

Here is an example of a complex Logo line. It is part of a procedure ONEUP that illustrates the use of the operation BUTLAST in Chapter 7, “Logo Primitives.”

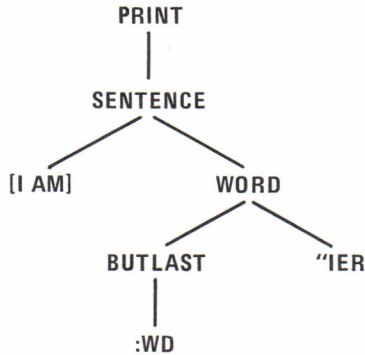
```
PRINT SENTENCE [I AM] WORD BUTLAST :WD “IER
```

Let us see how our guidelines help in splitting the line.

PRINT is a command with a single input. This input must be SENTENCE, which is an operation with two inputs.

The first is the list [I AM]. The second is the operation WORD. The latter is, again, an operation with two inputs. The first must be the operation BUTLAST, which has a single input :WD. The second input to WORD must, therefore, be “IER.

Because there are no more procedure names, every input on the line has been accounted for. The following diagram summarizes what we have done.



Words, Lists, and Special Characters

You can make up a word using almost any character on the keyboard, including numbers and punctuation marks.

HELLO
X
314
3.14
PIGLATIN
PIG.LATIN
HEN3RY
WHO?
!WOW!

However, you cannot use these characters:

- Space (Remember this is a character even though you cannot see it.)
- Left [and Right] brackets
- Left (and Right) parentheses
- Plus + and Minus -
- Asterisk * and Backslash \
- Equals =, Greater-than >, Less-than <

When Logo sees one of these characters it thinks a new word is about to begin. Because of this, these characters are called *delimiters*. We are already familiar with how the space does this. The other characters work in the same way to separate words. There is no need to put a space between a word and any of these characters. For example, the line

```
IF 1<2[PRINT(3+4)/5][PRINT :X+6]
```

is understood by Logo exactly like

```
IF 1 < 2 [PRINT (3 + 4) / 5] [PRINT :X →
+ 6]
```

As you may have noticed, Logo also inserts spaces. If you define a procedure to contain a line in the first form, then run the procedure, Logo converts it into the second.

Note that the quote and colon are not on this list. They are special characters only if they occur at the *beginning* of a word. If they occur in the middle or at the end, they are part of an ordinary word.

For example,

JOHN“MARY” is one word.

JOHN(MARY) is two words.

JOHN:MARY is one word.

You can change a delimiter into an ordinary character by typing a backslash before it.

For example:

?PRINT "SAN\ FRANCISCO

SAN FRANCISCO (a single word containing a space)

Left and right brackets always indicate the beginning and end of a list or sublist.

[HELLO THERE, OLD CHAP]

[X Y Z]

[HELLO]

[[HOUSE MAISON] [WINDOW FENETRE] [DOG CHIEN]]

[HAL [QRZ] [BELINDA MOORE]]

[1 [1 2] [17 2]]

The list [HELLO THERE, OLD CHAP] contains four elements:

Element 1: HELLO

Element 2: THERE,

Element 3: OLD

Element 4: CHAP


Note that the list `[1 [1 2] [17 2]]` contains only three elements, not six. The second and third elements also are lists:

Element 1: 1

Element 2: `[1 2]`

Element 3: `[17 2]`

The list `[]`, a list with no elements, is an *empty list*. There also exists an *empty word*, which is a word with no elements. You type in the empty word by typing a quotation mark followed by a space. See the EQUALP operation in Chapter 7 for examples of both the empty list and the empty word.

Note that Logo will end a list for you automatically. If the end of a Logo line is reached (the Enter  key is pressed and there are not enough brackets to close a list), Logo will automatically put them in at the end of the line.

```
?REPEAT 4 [PRINT [THIS [IS [A [TEST
THIS [IS [A [TEST]]]
THIS [IS [A [TEST]]]
THIS [IS [A [TEST]]]
THIS [IS [A [TEST]]]
```

If Logo finds a right bracket without a corresponding left bracket, Logo ignores the right bracket.

Chapter 3. Logo Editor

Contents

EDIT Command	3-3
How the Editor Works	3-4
Editing Actions	3-5
Cursor Motion	3-5
Inserting and Deleting	3-7
Scrolling the Screen	3-8
Exiting from the Editor	3-8
Outside the Editor	3-9
Using the Editor	3-9

This chapter describes how to define and change Logo procedures by using the EDIT command, and explains each editor action in detail.

EDIT Command

EDIT is the command used to start the Logo editor. Its input is the name of the procedure to be edited. If a procedure is already defined, its definition is brought into the editor, where you can modify it. If a procedure name is undefined, the editor shows only the title line

TO SQUARE ■

so you can define it.

The input to EDIT can be a list of procedure names instead of a single name. In this case, all procedure definitions will be brought into the editor. For example:

EDIT [HOUSE ROOF DOOR]

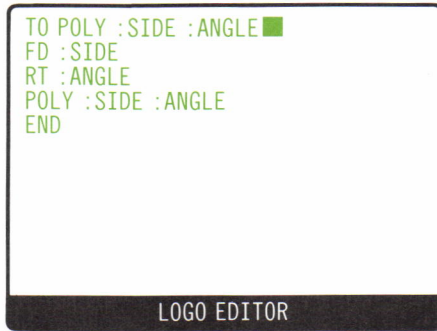
will bring in all three procedures.

EDIT can be used without an input. In this case, the screen shows the editor as it was when last used. If you haven't used the editor since loading the Logo language, the editor displays a blank page. You can start defining a procedure by typing TO and the procedure name.

How the Editor Works

When the editor is called, the screen changes. For example:


?EDIT "POLY




There is no prompt character but the Cursor ■ shows where you are typing.


The text that you edit is in an area in memory called a *buffer*. When you enter the Logo editor, the text from the buffer is displayed, up to 24 lines per screen.

You can move the cursor anywhere on the text by using the cursor control keys described in this chapter. Move the cursor to insert new characters or delete existing ones.

Each key typed makes the editor take some action. The typewriter characters (letters, numbers, punctuation, and Enter ) are inserted into the buffer at the place marked on the screen by the cursor.

The cursor keys, function keys, and some Ctrl key combinations have special meanings to help you edit.

Pressing the Enter  key marks the end of the Logo line. The cursor, and any text that comes after it, moves to the beginning of the next line and is ready for you to continue typing.

You can have more characters on a Logo line of text than fit across the screen. To do this, continue typing when you get to the end of the line. Do not press the Enter  key. An → (arrow) will appear at the end of the line, and the Cursor ■ will move to the next line.

Logo does the same thing outside of the editor. Such a line would appear like this on the screen:


```
TO PRINTMESSAGES :PERSON
PRINT SENTENCE :PERSON [, I AM GOING TO→
  TYPE A VERY LONG MESSAGE FOR YOU]
PRINT SENTENCE [SO LONG,] :PERSON
END
```


The editor has a line buffer called the *delete buffer*. It can be used to move lines in a procedure or to repeat them in different places. This buffer can hold a maximum of 128 characters. When a line of text is deleted, it can be put into this buffer so you can insert it at another location.

Editing Actions

When in the editor, you can use the following editing actions: cursor motion, inserting and deleting, and scrolling.

Cursor Motion

 Cursor Right moves the cursor one space forward (to the right).

 Cursor Left moves the cursor one space backward (to the left).



Cursor Down moves the cursor down to the next Logo line. The cursor tries to go to the character position directly beneath its position on the current line. If the next line is shorter than the cursor position on the current line, the cursor goes to the end of the next line. If the cursor is on the last line of the buffer, it does not move and you hear a beep.

Examples:

THIS IS A TEXT LINE

Cursor on L

THIS IS ANOTHER TEXT LINE

Cursor on space

A SHORT ONE

Cursor on end of line

THIS IS A LONGER ONE THAN CAN FIT ON THE SCREEN



Cursor on R

THIS IS THE NEXT LINE



Cursor on T





Cursor Up moves the cursor up to the previous line. The cursor tries to go to the character position directly above its position on the current line. If the previous line is shorter than the cursor position on the current line, the cursor moves to the end of the previous line. If the cursor is on the first line of the buffer, it does not move and you hear a beep.


 **←** (Beginning of line) moves the cursor to the beginning of the current line. Press this key while holding down the Shift  key.


→ (End of line) moves the cursor to the end of the current line (to the right of the last character on the line).


  (End of buffer) moves the cursor to the end of the text in the edit buffer.



  (Beginning of buffer) moves the cursor to the beginning of the edit buffer.



Inserting and Deleting



 The Enter key creates a new line at the current cursor position and moves the cursor to the beginning of the new line.


 The Insert key opens a new line at the position of the cursor but does not move the cursor.

 The Backspace key (large gray left arrow key at top right of keyboard) erases the character to the left of the cursor and the cursor backs up one position.

 Delete erases the character *at* the cursor position. Compare with Backspace  key.

  (Delete to end of line) deletes text from the cursor position to the end of the current line. This text is placed in the delete buffer, which can hold up to 128 characters.

  (Insert last line deleted) inserts, at the current cursor position, a copy of the text that is in the delete buffer.

 Function key 3 copies, at the current cursor position, the last line you typed.

Scrolling the Screen

If the text in your edit buffer has too many lines to fit on the screen, use these commands to control what is shown on the screen. Changing the portion of the buffer shown is called *scrolling* because you can imagine the buffer being rolled up and down past the screen.

3 PgDn Page Down scrolls forward one page. The cursor is moved to the beginning of the first line of the new page on the screen. If there is no new page of text beyond the screen, the editor beeps and does nothing.

9 PgUp Page Up scrolls back one page. The cursor moves to the beginning of the first line of the previous page on the screen. If there is no previous page of text, the editor beeps and does nothing.

Exiting From the Editor

Esc Escape is the standard way to exit from the editor.












When you exit from the editor by pressing the **Esc** key, Logo reads each line in the edit buffer. The procedures that have been defined will now be part of your workspace.

If there is a title line (TO . . .) in the edit buffer which starts the definition, Logo behaves as though you had typed the definition of the procedure by using the command TO. If the buffer contains one procedure definition but you forgot to type the END instruction at the end of the definition, Logo inserts END for you. You can define many procedures while in the editor, as long as the instruction END closes each procedure. In this case, if you omit END from the final procedure definition, Logo will insert it for you.

If there are Logo instructions in the edit buffer that are not contained in the procedure definition (within TO . . . END), Logo carries them out as you exit from the editor as if you had typed them in at top level, outside the editor.

Ctrl-Break Ctrl-Break stops the editing. Use it if you decide not to make changes or dislike the ones made. These changes will be ignored. If you were defining a procedure, the definition will be the same as when you started editing. If you press Ctrl-Break by mistake, you can get back to the edit buffer by typing EDIT (no input).

Outside The Editor

When editing a single line, use these keys: Cursor Right , Cursor Left , Beginning of line , End of Line , Backspace , Delete , Delete to end of line  - , Insert last line deleted  - , and Function key 3 .

Using the Editor

The following example defines a procedure in which you have to guess the number that the computer has chosen. Enter the editor and type:

```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT [WRONG]]
END
```

Press  to exit from the editor.

When you try the procedure, this may result:

```
?GUESSNUM  
GUESS A NUMBER  
5  
WRONG  
?
```

If you want to change this procedure so it tells you the number if you are wrong and starts over, the new procedure should look like this:

```
TO GUESSNUM  
MAKE "NUMBER RANDOM 10  
PRINT [GUESS A NUMBER]  
MAKE "GUESS READWORD  
TEST :GUESS = :NUMBER  
IFTRUE [PRINT [RIGHT]]  
IFFALSE [PRINT SE [WRONG, THE NUMBER IS→  
] :NUMBER]  
GUESSNUM  
END
```

When you try the procedure, you may see:

```
?GUESSNUM  
GUESS A NUMBER  
7  
WRONG, THE NUMBER IS 3  
GUESS A NUMBER  
6  
RIGHT  
GUESS A NUMBER  
.  
.  
.
```



Let's edit our old procedure and change it to the new one.

?EDIT "GUESSNUM

The Logo editor screen shows:

```
TO GUESSNUM ■
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT [WRONG]]
END
```

LOGO EDITOR

Now use Cursor Down  to move the cursor down to the seventh line and Cursor Right  to move the cursor onto the [(Left Bracket) to the left of WRONG.

```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT [WRONG]]
END
```

To add SE where the cursor is now type SE and press the spacebar. Notice that the text on the line moves out of the way.

```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT SE [WRONG]]
END
```

The cursor should still be on the bracket. Press End of Line → to move the cursor to the end of the line. Now press Backspace ← twice to erase the two] (Right Brackets) to the left of the cursor. Type the rest of the line so it looks like this:

```
IFFALSE [PRINT SE [WRONG, THE NUMBER IS→
] :NUMBER]
```

Press Enter ↵ at the end of the line to put the cursor at the beginning of an empty line.

```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT SE [WRONG, THE NUMBER IS→
] :NUMBER]
█
END
```

Type

```
GUESSNUM
```


Now press the **Esc** key to exit from the editor. The message **GUESSNUM DEFINED** appears on the screen. Try your new **GUESSNUM** procedure.








The new procedure is recursive. To stop the procedure, press Ctrl-Break.

Get back into the editor to see a few other editing techniques. Type







EDIT



Move the cursor to the beginning of the third line by pressing Cursor Down **↓**. Press **Ctrl** - **↵**. This means delete the rest of the current line. You will see that line disappear. You can bring the line back by pressing **Ctrl** - **↶**. This means insert the delete buffer. Try it.


When you press **Ctrl** - **↵**, the editor puts the text on the rest of the current line into the delete buffer and then erases that text from the screen. The text stays in the delete buffer. You can then insert that text at the position of the cursor (as if you typed it again) by pressing **Ctrl** - **↶**. This can be used for moving lines in a procedure or for putting in several copies of a line.

Try this. Move the cursor to the beginning of the last line by pressing Cursor Down  or Cursor Right . Press the  key. The editor action means Insert a line. Now press  - . This inserts the text from the delete buffer again. Press the Enter  key. Press function key . This puts another copy of that text at the position of the cursor.




```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUM
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT SE [WRONG, THE NUMBER IS→
] :NUMBER
GUESSNUM
PRINT [GUESS A NUMBER]
PRINT [GUESS A NUMBER] ■
END
```

Move some lines by using ,  - ,  - , and . Don't worry about ruining the GUESSNUM procedure.

Another editing action occurs when you press the  key. This deletes the character *at the current cursor position*. The Backspace  key (large gray left arrow key at top right of keyboard), in contrast, deletes the character *to the left of the current cursor position*. Both of these actions are useful when you type.

Move the cursor to various places on the screen and press the  key. The character under the cursor is deleted.

This is useful for correcting typing mistakes made earlier. Move the cursor onto the wrong characters and delete them.

What happens if you move the cursor to the end of a line (use →) and press the  key? Try it. The next line “moves up” and merges with the current line because you deleted the carriage return produced by pressing the Enter  key at the end of the line you were on. Put it back by pressing the Enter  key.

Now you know the fundamentals of using the Logo editor. You’ve also probably ruined the appearance of the GUESSNUM definition on the screen. If so, and you want to get back to Logo *without* keeping the messy definition, exit from the editor by pressing Ctrl-Break. This gets you out of the editor but gives no new information to Logo. GUESSNUM is not redefined. Now type

```
PO "GUESSNUM
```

Logo will print the last definition of GUESSNUM.

```
TO GUESSNUM
MAKE "NUMBER RANDOM 10
PRINT [GUESS A NUMBER]
MAKE "GUESS READWORD
TEST :GUESS = :NUMBER
IFTRUE [PRINT [RIGHT]]
IFFALSE [PRINT SE [WRONG, THE NUMBER IS→
] :NUMBER]
GUESSNUM
END
```


Chapter 4. File Handling

Contents

Naming Files	4-3
Disk File Names	4-3
Device Names	4-5
Permanent and Temporary Storage	4-6
Types of Files	4-6
Directory File	4-8
Program File	4-8
Organizing Your Procedures in Files	4-9
Screen File	4-11
Dribble File	4-12
Data File	4-13
Step 1: Creating a Data File	4-14
Step 2: Retrieving Information	4-17
Step 3: Changing Information	4-18
Binary File	4-19
Printing Files	4-20
Printing to the Printer	4-22

This chapter explains the Logo file handling facilities. A *file* is an organized collection of information that can be permanently stored for specific purposes.

Naming Files

Logo thinks of a file as a disk file or a system device, such as a printer or the screen. The input for a file primitive can be chosen by following the guidelines offered in the next two sections.

Disk File Names

The name of each file can be made up of three parts, called the file specification or *filespec*. The three parts are:

disk drive: *file name* *.ext*

The *disk drive* can be A: (the first drive on your system), B: (the second drive), or any other drive you have on your system. If you do not enter this input, the disk drive is assumed to be the default drive. You can change the default drive by using the SETDISK command.

The *file name* is the name you give to your disk file. This can be from 1 to 8 characters long. If the file name is more than 8 characters, Logo uses only the first 8. The file name can be followed by an extension.

The following characters can be used when creating file names:

A-Z 0-9 \$ & # @ ! % ' { } _ ~ ^ ,

Other characters have special meanings in Logo.

The *.ext* (extension) consists of a period and from 1 to 3 characters. When used, the extension immediately follows the file name. There are no spaces between the file name and the extension. Logo automatically adds the extension *.LF* to program files that are inputs to **LOAD**, **SAVE**, and **ERASEFILE** and adds *.PIC* to screen files that are inputs to **LOADPIC** and **SAVEPIC**. In order to avoid these automatic extensions, you *must* add a *.* (period) to the end of the file name. If you have an extension for any other kind of file, include the extension when you refer to that file. Otherwise, Logo won't be able to locate it.

These characters can be used for extensions:

A-Z 0-9 \$ & # @ ! % ' { } _ ~ ^ ,

Other characters have special meanings in Logo.

For example,

```
SAVE "C:MARY
```

saves the file called **MARY.LF** onto the disk in drive C.

```
OPEN "B:DATAFILE.DAT
```

opens the file called **DATAFILE.DAT** on the diskette in drive B.

```
SAVEPIC "SCENE
```

Saves the screen picture into a file called **SCENE.PIC** on the disk in the default drive.

Device Names

You can use a device name as input to many file primitives. These are names reserved for system devices along with associated devices and read/write status.

Reserved Name	Device	Read/Write
CON	Keyboard/screen	Both
AUX or COM1	First Asynchronous Communications Adapter	Only for writing
COM2	Second Asynchronous Communications Adapter	Only for writing
LPT1 or PRN	First Parallel Printer	Only for writing
LPT2 or LPT3	Second Parallel Printer Third Parallel Printer	Only for writing
NUL	Nonexistent (dummy) device	Testing applications; simulates writing only.

For example,

```
SAVE "LPT1
```

prints the workspace onto the printer.

```
OPEN "AUX
```

Opens the First Asynchronous Communications Adapter.

Permanent and Temporary Storage

When you define a procedure and run it to manipulate various data, Logo allocates temporary workspace in the computer memory. Refer to the memory map in Appendix C for the exact size of your workspace. When you switch off the computer, leave Logo, or restart, the contents of this space are erased.

To save information for future use you must store it on a permanent storage medium, which can be a disk or a printout on paper. The advantage of a disk is that information stored on it can be accessed directly by the computer when needed.

Types of Files

Logo creates different types of files according to the nature of the information to be stored.

File	Contents
Directory File	Contains the table of contents of all files on a disk. This is a special type of file and is created only by the Disk Operating System (DOS).
Program File	Contains a copy of procedure definitions, names, and properties used in the workspace.
Screen File	Contains an image of the display on the screen when the file is created.
Dribble File	Contains a copy of almost all text that goes to the screen, including the instructions typed at the keyboard and the computer's response to it. A few characters cannot be dribbled; see the "Dribble File" section of this chapter for details.
Data File	Contains a sequence of characters, words, and lists that are stored to be retrieved later.
Binary File	Contains an unrestricted sequence of bytes—most useful for storage of machine-code routines and raw data.

Directory File

A special area is reserved on each disk for storing its directory (or table of contents). This file is created and managed by DOS and cannot be modified explicitly by Logo commands.

The Logo command **DIR** lists the contents of the directory file on the screen. It shows the file name, its extension (or file type), the file size in bytes, and the most recent date and time that the file was altered or created. For more information, see the **DIR** command in Chapter 7.

Program File

The procedures defined in the workspace can be stored on the disk by the command **SAVE** *filespec*. **SAVE**, with a given file name as the input, creates this file and puts in it a copy of the entire contents of the workspace.

Everything in the workspace does not have to be saved. If you do not want the names, or variables, created by the command **MAKE** to be stored in a program file, you can erase all of the names. To do this, use the command **ERNS** before you save your procedures. If any procedures are not worth saving, they can be erased by using the command **ERASE** with their procedure names as input. See Chapter 4 of *Programming with Turtle Graphics* for more information on workspace management.

Organizing Your Procedures in Files

You can classify your procedures into separate files in the workspace. Let's say you have ten procedures jumbled up in your workspace—three for computer poems, four for an arithmetic quiz, and three for turtle graphics. You want to save the three procedures of poems without saving the other procedures. To make this bookkeeping easy, Logo provides a primitive called **PACKAGE**. You can group your procedures and names by putting them in a specific package. For example,

```
PACKAGE "POEMS [NOUNS VERBS COMPOSE]
```

puts the procedures named **NOUNS**, **VERBS**, and **COMPOSE** into the package named **POEMS**. When this is done, you can file these three procedures under the name **POETRY** by using the command

```
SAVE "POETRY "POEMS
```

If the three procedures are in your workspace when you enter this command, Logo puts them into the file **POETRY** and responds

```
3 PROCEDURES SAVED
```

You may want to package another set of procedures.

```
PACKAGE "QUIZ [ADD PLAY MULTIPLY RESUL  
T]
```

You can save both packages in the same file.

```
SAVE "POETRY.QIZ [POEMS QUIZ]
```

Logo responds

7 PROCEDURES SAVED

You can continue to classify the remaining procedures into other packages and create a file to save each package. If you want to store the rest in one file, you should erase the two packages and save the rest.

ERALL [POEMS QUIZ]
SAVE "GRAPHICS

Logo responds

3 PROCEDURES SAVED

To bring back the procedures stored in the file POETRY.LF, type

LOAD "POETRY

You can now edit your procedures and save them again by using

ERASEFILE "POETRY

to erase the old POETRY.LF file and

SAVE "POETRY "POEMS

to save the POEMS package in the new POETRY.LF file.

If you want to add a new procedure (for example, RHYMES) to the POEMS package before you save the procedures in the new POETRY file, you must type

PACKAGE "POEMS "RHYMES

before typing

SAVE "POETRY "POEMS

You do not have to repackage the other procedures in the POEMS package. Check this by typing

POTS "POEMS

Logo prints

TO RHYMES :WORDS

TO COMPOSE :NOUNS :VERBS

TO NOUNS

TO VERBS

Screen File

A screen file saves the contents of a screen (graphics, text, or both) in a file.

If you want to save the turtle's screen drawing instead of saving the procedures used to draw it, use the command SAVEPIC with a quoted file name.

Suppose you have created a picture of a forest on the graphics screen. While that picture is displayed on the screen, type

SAVEPIC "FOREST

This command creates a file named FOREST.PIC on a disk. The picture currently displayed on the screen is saved in this file. The directory display produced by the command DIR shows the extension .PIC immediately after the file name.

Once the picture is saved in a file, you bring it back to the screen (blank or otherwise) with the command **LOADPIC**, followed by the specific file name. Therefore,

```
CS  
LOADPIC "FOREST"
```

brings back the previous image of the forest after the graphics screen is cleared by CS.

Dribble File

A dribble file can contain a copy of most of the characters displayed on the text screen. It gives a record of the interactions between the computer and the person at the keyboard. For example, teachers can use the information stored in a dribble file to analyze the learning process that took place in a course. This can help a teacher understand the bugs encountered by students and how they dealt with them.

A dribble file can be written on a disk or printed on paper by using a printer.

If you direct your dribble file to an IBM 80 CPS Matrix Printer, the following characters can be printed:

ASCII codes	033 - 126, and 161 - 223.
-------------	------------------------------

If you direct your dribble file to a disk file, then you may dribble the following characters:

ASCII codes	000 - 025, and 027 - 255.
-------------	------------------------------

Two primitives, DRIBBLE and NODRIBBLE, turn the dribble feature on and off. For example,

DRIBBLE "MAY6

creates a file named MAY6, which stands for the session on the 6th of May. After the dribble feature is turned on, every line appearing on the text screen will be saved in this file until dribble is turned off by using the NODRIBBLE command.

NODRIBBLE is the command that stops dribbling and closes the dribble file. If you switch off the computer before entering this command, the information in the buffer may be lost.

Data File

Characters, words, and lists of words can be named and stored as variables in a program file. This is convenient as long as a small quantity of information is used.

For dealing with a large quantity of information, you need a facility of file management, which will give you direct access to the contents of a file.

For example, if you want to store a telephone directory of 100 members of a social club, it would be a good idea to organize the telephone numbers into a data file. Then, when you want to find a number, you can pick up only the information needed rather than loading the 100 numbers all at once.

Let's examine the file management system using this telephone directory project as an example. The objectives of this project are:

- To store the members' names and their phone numbers
- To find out a particular member's phone number
- To change a member's phone number

Step 1: Creating a Data File

Here is a procedure that reads the name and the phone number of a member interactively from the keyboard:

```
TO ASKINFO
PRINT [TYPE IN THE MEMBER'S NAME:]
MAKE "NAME READLIST
PRINT [TYPE IN THE PHONE NUMBER:]
MAKE "TEL READLIST
END
```

ASKINFO prints the message on the screen, takes the answer from the keyboard, and gives a name to this answer. When ASKINFO finishes its job, it creates two variables: NAME and TEL. The next step is to write the information into a file.

Logo treats a file as if it were another device. Just as it can print a message on the screen, it can print it in a file. It can pick up information from a file with READLIST just as it did from the keyboard.

A data file can be written to a printer or a disk. We only need to specify where to write or read. SETWRITE is the command that specifies the writing device.

```
TO WRITEINFO
```

```
SETWRITE "MEMBERS
```

MEMBERS is the file name.

```
PRINT :NAME
```

```
PRINT :TEL
```

```
SETWRITE "CON
```

CON stands for CONsole. This command directs the writing back to the screen.

```
END
```

The last step is to write the superprocedure to open the data file called MEMBERS, run these subprocedures, and close the data file.

```
TO SAVEINFO
```

```
OPEN "MEMBERS
```

```
ASKINFO
```

```
WRITEINFO
```

```
CLOSE "MEMBERS
```

```
END
```

Now try the procedure:

```
?SAVEINFO
```

```
TYPE IN THE MEMBER'S NAME:
```

```
SHEILA O'NEIL
```

```
TYPE IN THE PHONE NUMBER:
```

```
555-5800
```

```
?
```

The program finished running, but you can't see what happened to the data file. To check the result, print out the file.

```
?POFILE "MEMBERS
```

The screen displays everything written in the data file **MEMBERS**.

```
SHEILA O'NEIL  
555 - 5800
```

What happens if we run the procedure again?

```
?SAVEINFO  
TYPE IN THE MEMBER'S NAME:  
GUY GROEN  
TYPE IN THE PHONE NUMBER:  
555-1563  
?
```

SAVEINFO worked just like it did the first time. Now look at the result.

```
?POFILE "MEMBERS  
SHEILA O'NEIL  
555 - 5800  
GUY GROEN  
555 - 1563  
?
```

The procedures work for adding more members as well as for creating the data file for the first time.

Step 2: Retrieving Information

After creating the data file containing names and phone numbers, the next step is to build a program to find a particular member's phone number.

```
TO FINDINFO
PRINT [TYPE IN THE MEMBER'S NAME:]
MAKE "NAME READLIST
OPEN "MEMBERS
SETREAD "MEMBERS
FINDTEL :NAME
CLOSE "MEMBERS
END

TO FINDTEL :NAME
IF RL = :NAME [PR SE [THE PHONE NUMBER →
IS:] RL STOP]
IF READEOF [PR [CAN'T FIND THIS NAME] →
STOP]
FINDTEL :NAME
END
```

FINDINFO is the superprocedure. First, it reads from the keyboard the name of the person whose phone number is wanted.

Then, it opens the data file and tells Logo that it wants to read information from this data file.

The subprocedure FINDTEL starts reading line by line from the beginning of this data file using READLIST (RL). Each time it reads a line, FINDTEL compares it with the name it is looking for. If they are identical, it reads another line and prints the sentence

THE PHONE NUMBER IS:

If not, it checks to see if READLIST has reached the end-of-file position (READOFP). If the end-of-file position has been reached, FINDTEL prints the message:

```
CAN'T FIND THIS NAME
```

Otherwise, it continues to read the next line.

Step 3: Changing Information

A member's phone number may change. So, we must be able to update the data file.

In order to modify a part of the data, we must know the location of the information to be changed. The procedures to retrieve the information (FINDINFO and FINDTEL) can be used for this purpose. Once the location is found, we can write the procedure MODIFY, which rewrites the information at this location.

```
TO MODIFY :LOCATION  
  PRINT [TYPE IN THE NEW PHONE NUMBER]  
  SETREAD "CON  
  SETWRITE "MEMBERS  
  SETWRITEPOS :LOCATION  
  PRINT READLIST  
END
```

The command SETREAD "CON tells Logo that we want to read the data from the keyboard. SETWRITE "MEMBERS tells Logo that we want to direct the next PRINT command to write the new data into this file. The command SETWRITEPOS :LOCATION makes sure that it is written at the current location.

Thus, the last command PRINT READLIST picks up the data from the keyboard and prints it into the file.

Now we must incorporate this procedure into the FINDTEL procedure. FINDTEL will read the file line by line, comparing each line to the name it is looking for. It will then call MODIFY with the LOCATION it gets from READPOS in the procedure FINDTEL. READPOS is the input to MODIFY. In addition to changing FINDTEL, change the name of the superprocedure FINDINFO to MODINFO.

```

TO MODINFO
PRINT [TYPE IN THE MEMBER'S NAME:]
MAKE "NAME READLIST
OPEN "MEMBERS
SETREAD "MEMBERS
FINDTEL :NAME
CLOSE "MEMBERS
END

TO FINDTEL :NAME
IF RL = :NAME [MODIFY READPOS STOP]
IF READEOFP [PR [CAN'T FIND THE NAME] →
STOP]
FINDTEL :NAME
END

```

Binary File

Not all data exists as printable characters. Examples are the sequences of bits used to define the patterns on the screen we recognize as characters and the assembly language routines through which all instructions eventually direct the microprocessor.

Binary files are a more general class of data files that may include *any* bytes, without restriction. Attempting to print a binary file directly onto the screen or the printer is usually not worth the effort because the control characters ensure unreadable results.

Logo interacts with this type of file through several of the primitives beginning with a . (period). An example is .BLOAD, which is used to bring a binary file into computer memory from disk. If you need to know more about binary files, see the definitions of .BLOAD, .BSAVE, .CALL, .DEPOSIT, and .EXAMINE in Chapter 7, “Logo Primitives” and the information in Appendix C, “How Logo Uses Memory.”

Printing Files

Logo thinks of a printer as another kind of file. Just as it can save your procedures in a file, Logo can save them on a printer. LPT1 or PRN is the device name for a parallel printer. COM1 or AUX is the device name for a serial printer. Before you can use a serial printer, you *must* use .SETCOM with the appropriate parameters for your serial printer.

The following command causes the printer to print all procedures, names, and properties in the workspace:

```
SAVE "LPT1
```

SAVE can receive a second input, the package name, in this context as well. So,

```
SAVE "LPT1 "POEMS
```

causes the four procedures in the POEMS package to be printed on the printer.

The only difference between a file on a disk and a printout from a printer is that Logo cannot read information printed on a printer. That makes

LOAD "LPT1

an impossible command for Logo to carry out.

Directing DRIBBLE to the printer instead of to a file allows you to print a dribble file.





DRIBBLE "LPT1

starts dribble on the printer. If you want to print the contents of an already created file on the printer, you can use the following procedure:

```
TO DUMP :FILENAME
DRIBBLE "LPT1
POFILE :FILENAME
NODRIBBLE
END
```

You also can print the contents of your screen on a printer. If you have an IBM Personal Computer Graphics Printer connected and have a picture on the screen, type

SAVEPIC "LPT1

You can achieve the same effect by pressing the Shift  - Print Screen  key combination to print your picture if you are working with a TV or monitor. The Shift  -  key combination will not print your graphics screen if .SCREEN is 2 (that is, you are working with two screens).

Printing to the Printer

Usually the screen is the destination of your print statements. To change the destination so that any information printed appears on the printer instead of on the screen, type

```
TO SCRIBBLE
OPEN "LPT1
SETWRITE "LPT1
PRINT [THIS LINE WILL BE PRINTED ON PAP→
ER]
SETWRITE "CON
PRINT [THIS WILL BE DISPLAYED ON THE SC→
REEN]
SETWRITE "LPT1
PRINT [THIS LINE GOES TO THE PRINTER]
PRINT [HARDCOPY IS PRINTED ON PAPER]
CLOSE "LPT1
END
```

```
?SCRIBBLE
THIS WILL BE DISPLAYED ON THE SCREEN
```

The other lines are printed on paper.

Chapter 5. How Logo Handles Numbers and Mathematical Operations

Contents

Numbers that Logo Recognizes	5-5
Floating-point Format Examples	5-5
Examples of Exponential Format of Numbers and the Equivalent Scientific Notation	5-8
How Logo Outputs Numbers	5-9
PRECISION and Logo Numbers	5-13
Order of Mathematical Operations	5-16
Special Prefix Operations	5-17
The — (Minus) Sign	5-18
Turtle Angles and Math Angles	5-19

In Logo, numbers are treated as literal words but words of a special kind—words that needn't be quoted and that can be inputs to arithmetic and other operations of mathematics. Mathematically, Logo numbers are real numbers that can have up to 1,000 places.

This chapter describes:

- The forms of numbers that Logo recognizes.
- The forms numbers take when Logo outputs them.
- How to change the precision of Logo numbers.
- The order in which arithmetic and other math operations are executed.
- Infix operations that can be used as special prefix operations.
- How Logo interprets the $-$ (minus) sign.
- How turtle-heading angles differ from angles used by some math operations.

Logo numbers are words that don't have to be quoted.
For example,

```
?PR WORDP "NUMBERS
```

```
TRUE
```

```
?PR WORDP 12.5
```

```
TRUE
```

```
?PR "NUMBERS
```

```
NUMBERS
```

```
?PR 123.2
```

```
123.2
```

```
?PR COUNT "NUMBERS
```

```
7
```

```
?PR COUNT 1234
```

```
4
```

```
?PR WORD "NUM "BERS
```

```
NUMBERS
```

```
?PR WORD 12 3.5
```

```
123.5
```

But numbers are special words and differ from words
consisting of non-numeric characters.

```
?PR NUMBERP "NUMBERS
```

```
FALSE
```

```
?PR NUMBERP 123.5
```

```
TRUE
```

```
?PR SUM "NUM "BERS
```

```
SUM DOESN'T LIKE NUM AS INPUT
```

```
?PR SUM 12.3 10.2
```

```
22.5
```

Numbers that Logo Recognizes

Numbers can be entered into Logo in either of two formats: floating-point or exponential.

Floating-point Format Examples

113	0.0	232.
15.2	323.0	-5.2
-16.03	-6.0	-3.
-.04	24.3	-6
12345.6789	0	232.24

Floating-point numbers consist of:

- An optional - (minus) sign.
- An integer part which need not be present if its value is zero but which otherwise consists of various digits.
- A decimal point, needed only if the number has a fractional part.
- A fractional part which need not be present if its value is zero but which otherwise consists of various digits.

There must be at least one digit in a number. Remember that blanks or spaces are word separators and cannot be used as or within numbers.

Note that +2, +2.5, and +0.03 are not recognizable forms of numbers for Logo. Logo always interprets the + (plus) sign as the addition operation and tries to perform this operation whenever it is encountered. This results in the following sorts of errors when the + is used to indicate positive numbers:

```
?PR + 5
NOT ENOUGH INPUTS TO +
?MAKE "A +2
+ DOESN'T LIKE A AS INPUT
?(PR 2 +5 -3)
7 -3
```

There are no such problems with the — (minus) sign. When a — immediately precedes a number, it is recognized as indicating a negative number.

```
?(PR 2 5 -3)
2 5 -3
?MAKE "A -12
?PR :A
-12
```

But if a space is inserted between the — (minus) sign and a number, the — is recognized as the infix subtraction operation.

```
?(PR 2 5 - 3)
2 2
```

The second form of numbers that Logo can handle is a special exponential form. This format consists of a number multiplied by some power of 10. For example,

$$5.137 \times 10^8$$

is used to represent 5.137 times 100,000,000 (10^8) or 513,700,000. This format, referred to as *scientific notation*, is very handy for comparing very large or very small numbers.

$$1.987 \times 10^{-4}$$

and

$$2.0314 \times 10^{-7}$$

Both represent very small numbers, but the scientific notation enables us to see quickly that the first number is about 1000 times larger than the second number since the powers of 10 differ by 3. This is not so readily apparent in their regular decimal forms, .0001987 and .00000020314. Scientific notation also enables us to avoid writing a lot of zeros.

Computers have been designed to make use of scientific notation in a variety of ways. They can recognize numbers in scientific notation and will often convert numbers into scientific notation. But, instead of using a number times 10 raised to a power, computers usually use a number followed by the letter E and an integer. The letter E is computer shorthand for the $\times 10$ part, and the following integer is the power. In strict scientific notation, which is what the computer always returns, the initial number part is always a number greater than or equal to 1 and less than 10. Logo handles numbers in scientific notation in this way.

Logo recognizes numbers as being in scientific notation if they consist of:

- A floating-point number between 1 and 10, including 1 (but not 10).
- The letter E. (E cannot be immediately preceded by a decimal point.)
- An optional + sign to indicate a positive exponent or a - sign to indicate a negative exponent.
- An integer from 0 to 9999, including 9999.

If numbers are entered in exponential format but not in strict scientific notation, Logo will recognize these numbers, convert them to scientific notation, and output them in its own scientific notation format. For example,

```
?PR -0.04321E60  
-4.321000000E+0058
```

Examples of Exponential Format of Numbers and the Equivalent Scientific Notation

Exponential	Scientific
1E2	1.0E2
1.235E4	1.235E4
21.63E-782	2.163E-781
0.4E0007	4.0E6

Note that blank spaces are not acceptable as or within a number. Note also that a decimal point is to be used only if the number has a fractional part.

For Logo to be able to deal with it, the exponent of a number must be in the range -9999 to 9999, inclusive, when the number is in scientific notation.

The number 99.9E9999 becomes 9.99E10000 which Logo does not recognize and, so, causes an error.

```
?PR 99.9E9999  
I DON'T KNOW HOW TO 99.9E9999
```

But .00001E10001 is recognized as a number:

```
?PR .00001E10001  
1.000000000E+9996
```

This worked because the exponent, 9996, is in the specified range.

Note that the E must always be both preceded and followed by a number. 1E2 represents the number 100 but Logo interprets E2 as merely the word E2 and not as a number.

```
?MAKE "A 1E2  
?PR :A  
100  
?MAKE "B E5  
I DON'T KNOW HOW TO E5
```

How Logo Outputs Numbers

Logo can output numbers in floating-point format or in scientific notation. While it prefers to use the floating-point format (as described above), it does output numbers in scientific notation when it has to output a number that has more places in it than a specified number of significant digits.

Logo rounds numbers to the number of significant places specified by the value of PRECISION. For example, if the current value of PRECISION is 5, 0.555555 is rounded to and output as 0.55556. On the other hand, the significant digits in 0.0000000000555555 would be rounded as above, but the number would be output in scientific notation, 5.5556E-0011.

Floating-point notation is used if the number to be output has, at most, a length of the number of digits specified by the value of PRECISION.

When the number to be output must contain more characters either before or after the decimal point than the value of PRECISION, Logo uses scientific notation. This scientific notation output format consists of:

- A space or a - (minus) sign.
- A non-zero digit—a digit from one through nine.
- A decimal point.
- A number of digits that is one less than the value of PRECISION.
- The letter E.
- A + (plus) or a - (minus) sign.
- Four digits specifying the exponent.

A number output in this format uses eight more characters than the number of significant digits specified by the value of **PRECISION**: the initial space or – (minus) sign, the decimal point, the letter E, the exponent's sign, and the four digits of the exponent.

Examples:

```
?PR 12345.0006789  
12345.00068
```

```
?PR -10.000123450006789  
-10.00012345
```

```
?PR 10000000125555.4321  
1.000000013E+0013
```

```
?PR .0000000000543219876  
5.432198760E-0011
```

```
?PR .567891234567  
0.5678912346
```

```
?PR .000891234567  
8.912345670E-0004
```

These numbers are output in the default format that Logo uses to output numbers when **PRECISION** is set to 10. If you want to change this format (for example, to print tables) use the primitives **FORM** and **EFORM**.

There are two situations in which Logo will not be able to output numbers: when the numbers are too big or too small for Logo to handle. A number is too big for Logo if, when in scientific notation, its exponent is greater than 9999—when it contains more than 10,000 places to the left of the decimal if written in decimal form. A number is too small for Logo if, in scientific notation, its exponent is less than -9999—when it contains 9999 or more zeroes to the right of the decimal if written in decimal form.

If Logo encounters a number that is too big or too small when it reads a line, it simply does not recognize it as a number and tries to treat it as a regular word. This will usually produce the

I DON'T KNOW HOW TO . . .

error message. Logo will take this word to be the name of a procedure and try to run it until it discovers that no such procedure has been defined (unless, of course, for some strange reason you have used this word as the name of a procedure). Then it will give the error message.

If, in the course of performing mathematical operations, a number that is too small for Logo is generated, Logo records a zero and the computation continues. If a number too big for Logo is generated, Logo gives the

NUMBER TOO BIG

error message.

For some mathematical functions, even though the result may be a Logo number, if a calculation generates a number which is too big, Logo gives this error.

PRECISION and Logo Numbers

From the previous sections it should be obvious that you can change the value of **PRECISION**, which is set to 10 when you start Logo. To change the value of **PRECISION**, use the **SETPRECISION** command. The precision cannot be set lower than 5 or higher than 1000.

When you use **SETPRECISION**, it does not change any number already in the workspace. No rounding takes place when the command is given. Rather, output from operations, the keyboard, or a file is rounded to the precision that you set before being read into the workspace. Also, output from the workspace to the editor, the screen, a printer, a file, or to another procedure is rounded to that same precision.

In other words, Logo remembers numbers according to the value of **PRECISION** that was set when those numbers were entered into the workspace. But, it outputs them according to the current value of **PRECISION**, and it rounds all new numbers entering the workspace according to the current value—which means this is the way these numbers will be remembered.

The following examples illustrate this.

```
?SETPRECISION 6
?MAKE "A6 12342.5
?(PR :A6 :A6 + :A6)
12342.5 24685
?SETPRECISION 5
?PR :A6
12343
```

The output of the second PRINT has been rounded to five digits. The value of :A6 in your workspace is still 12342.5, but there is no way to tell this while PRECISION is set to 5.

```
?MAKE "S5 :A6 + :A6  
?PR :A6 + :A6  
24686
```

The inputs to the + operation have been rounded to five digits. The sum that Logo actually performed was 12343 + 12343.

```
?MAKE "A5 :A6  
?MAKE "P5 :A6 + 0  
?SAVE "ARIT  
0 PROCEDURES SAVED  
?SETPRECISION 6  
?PRINT :A6  
12342.5
```

The value of :A6 has been the same in the workspace all along. Now that PRECISION is back to 6, it can be shown.

```
?PRINT :S5  
24686
```

The sum :A6 + :A6 was performed while PRECISION was 5. The result :S5 has not been modified by the change of precision.

```
?PRINT :A5  
12342.5
```

Although MAKE “A5 :A6 was carried out while PRECISION was 5, MAKE is a command (not an operation) assigning a new name to the value of A6 which was unchanged. In response to MAKE “A5 :A6, Logo put into A5 exactly what was in A6.

```
?PRINT :P5  
12343
```

But when Logo executed the line MAKE “P5 :A6 + 0, it first performed the + operation, whose output was rounded to five digits. Because PRECISION was set to 5 it assigned this output to the variable P5. Changing the precision to 6 did not change P5 and that value was printed.

```
?LOAD "ARIT  
?PR :A6  
12343
```

The value of A6 was written into the ARIT file when the current precision was 5. Therefore, what was actually written into ARIT is 12343 and that is what was read into your workspace when you entered the command LOAD “ARIT.

Note that the editor changes numbers according to the current precision. Set the precision to 10, the default precision. Define NUM as follows:

```
?TO NUM  
>PRINT 1E1  
>PRINT 123451234512  
>END
```

Now print it out.

```
?PO "NUM  
PRINT 10  
PRINT 1.234512345E+0011  
END
```


Important Suggestions

When working at various precisions, make sure you are at the highest precision used before storing your workspace in a file or writing data into a data file.

Also, when you load a file into your workspace or read from a data file, make sure the precision is set at the most precise (highest) number in the file to be loaded or read from.

Order of Mathematical Operations

When there are several math operations in a Logo line, Logo evaluates them according to the operations' precedence. The order of precedence from highest to lowest is as follows:

-	unary minus. Indicates a negative number (-3), or the additive inverse of its input. (-XCOR)
*, /	multiplication and division.
+, -	addition and subtraction.
Other math operations	(This includes user-defined as well as primitive operations such as SIN, DIFFERENCE and LN.)
>, <	greater than, less than
=	is equal to
AND, NOT, OR	logical operations
PRINT, SHOW	primitive commands
?PRINT 3 > 4 = 3 > 4 TRUE	

The highest precedence operation in this example is $>$.
So we have the same effect as

```
?PRINT "FALSE = "FALSE  
TRUE
```

You can modify this order with the use of parentheses. Logo follows the standard mathematical practice of performing operations enclosed in parentheses before others. If there are several operations in a set of parentheses, the above order of precedence is again in effect. For example,

```
?PR 2*4+8/4  
10  
?PR 2*(4+8/4)  
12  
?PR (2*4+8)/4  
4
```

In the second example, the parentheses tell Logo that $4+8/4$ is the second input to the $*$ operation. And $4+8/4$ is evaluated by first dividing 8 by 4 and adding the result, 2, to 4, giving 6. This 6 is then multiplied by the 2, giving the final result, 12.

In the third example, $2*4$ (giving 8) is added to 8 (giving 16), and that result is divided by 4 (giving 4).

Special Prefix Operations

The infix operations $=$, $<$, $>$, $+$, $-$, $*$ and $/$ may also be used in prefix form.

```
?PRINT * 4 5  
20  
  
?PRINT (+ (* 2 3) (* 4 5))  
26
```

If the expression is ambiguous, the infix form is assumed.

```
?PRINT + * 2 3 * 4 5  
29
```

The second $*$ is ambiguous, so it is assumed to be an infix operation, and the expression is parsed as follows:

```
(+ (* 2 (3 * 4)) 5)
```

The $-$ (Minus) Sign

The way in which the $-$ (minus) sign is parsed is a little strange. The problem is that this one character is used to represent three different things:

- It is used to indicate that a number is negative as in -3 .
- It is a procedure of one input, called *unary minus* that outputs the additive inverse of its input as in $-XCOR$ or $-(-3*5+9)$.
- It is a procedure of two inputs that outputs the difference between its first input and its second as in $7-3$ and $XCOR-ycor$.

Logo tries to be clever about this potential ambiguity and figure out which one applies using the following rules:

- If the $-$ sign immediately precedes a number and follows any delimiter except right parenthesis $)$, the number is parsed as a negative number. This accounts for the following behavior:

SUM 20 -20 is parsed as 20 plus negative 20
 $3*-4$ is parsed as 3 times negative 4

- If the $-$ sign immediately precedes a word or left parenthesis, (, and follows any delimiter except a right parenthesis, it is parsed as the unary minus procedure. If `:NUMS` is the list `[-5 5]`,

`-FIRST :NUMS` outputs 5

`-LAST : NUMS` outputs -5

`-((FIRST :NUMS) + 20)` outputs -15

- In all other cases, the $-$ sign is parsed as a infix (or prefix) procedure with two inputs. For example

`3-4` is parsed as 3 minus 4.

`3 - 4` is parsed exactly like the previous example.

`- 3 4` gives the same output as `3-4`.

Turtle Angles and Math Angles

Turtle-heading angles are measured like compass-heading angles with up being north, right being east, etc. Mathematically speaking, positive turtle-heading angles are measured *clockwise* from *north*—from an imaginary line going straight up from the turtle's current position.

The `ARCTAN` math operation outputs an angle which, if positive, is measured *counter-clockwise* from the positive *x-axis*.

The arctangent function normally takes one input and returns the angle whose tangent is equal to that input. The Logo `ARCTAN` operation works like this when it is used with a single input. But `ARCTAN` can also take two inputs. Here is one way to think about using `ARCTAN` with two inputs:

`ARCTAN 7 4`

outputs the angle measured counter-clockwise from the positive *x-axis* to a line passing through the origin (0, 0) and point (4, 7).

The origin is the point where the x-axis and y-axis meet. Point (4, 7) is the point four steps right of the origin and seven steps up. If PRECISION is set to 6, ARCTAN 7 4 outputs 60.2551 indicating that the angle labeled MA in Figure 1 is a little over 60 degrees.

TOWARDS [4 7]

outputs a heading for the turtle to turn to if it is to aim *toward* the point whose coordinates are 4 and 7. Of course, the angle that is output will depend on the turtle's current location. Assuming that the turtle is at its start-up position (at the origin), TOWARDS [4 7] outputs the angle measured clockwise from the positive y-axis to the line through the point. The output from TOWARDS, a graphics command, is independent of the current precision. All graphics commands output five significant digits. If PRECISION is 6, this output is 29.744 (the angle labeled TA in Figure 1) or almost 30 degrees.

In these cases both angles (the “math angle” output by ARCTAN and the “turtle angle” output by TOWARDS) indicate the same direction measured in different ways.

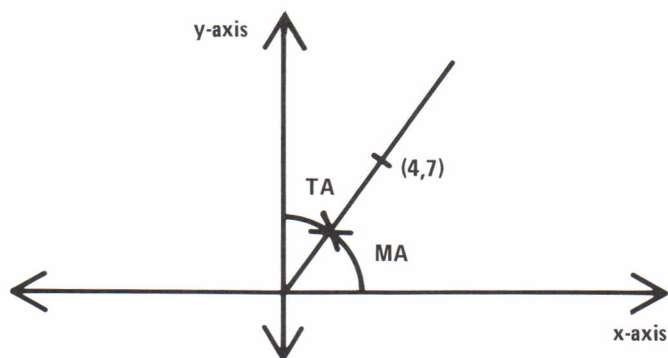


Figure 1

Another way to think about ARCTAN with two inputs is first to note that the tangent of an angle in a right triangle (like angle A in Figure 2) is equal to the length of the side opposite the angle divided by the side adjacent to the angle (9 divided by 7 in Figure 2). ARCTAN takes the length of these sides, does the division, and outputs the angle having that value as tangent. To find angle A in the triangle in figure 2 (if the precision is still 6)

ARCTAN 9 7

outputs 52.125. So angle A is approximately 52 degrees.

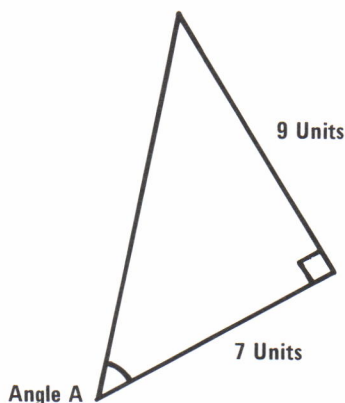


Figure 2

Chapter 6. Developing a Logo Project

Contents

The Subgoals	6-3
Step 1: Story Generator	6-4
Step 2: An Arithmetic Problem	6-6
Step 3: Variation in Problems	6-8
Suggestions for Modifications	6-11
Program Listing	6-12

This chapter shows you how to develop a project that deals with words, lists, and numbers. We will construct an interactive program that presents arithmetic “story” problems. Logo will print a problem, wait for an answer, and give immediate feedback on the answer. Here is a scenario:

MIMI HAD 10 CAKES
SHE BOUGHT 5
HOW MANY DID SHE HAVE THEN?
15
RIGHT!

JOE HAD 8 BOOKS
HE LOST 3
HOW MANY DID HE HAVE THEN?
11
GOOD TRY BUT THE ANSWER IS 5

To develop this program, our approach will be to divide the project into subgoals. We’ll do this by simplifying the task. We will need several steps to produce a version of the program that can generate the example. Each version of the program will be progressively closer to our final goal. This approach is a good model for developing Logo projects.

The Subgoals

Let’s take a look at the example. Each problem consists of two statements and a question.

MIMI HAD 10 CAKES
SHE BOUGHT 5
HOW MANY DID SHE HAVE THEN?

Logo waits for you to type a response.

15

Logo then responds to your answer.

RIGHT!

This program can be broken down into the following three steps:

- Step 1: Create a program that generates random stories.
- Step 2: Turn it into an arithmetic problem, making Logo pick up a response and give feedback on the solution. We will begin with addition problems.
- Step 3: Vary the type of problem so there are both addition and subtraction problems. Include male names and matching pronouns. This version will produce the example scenario.

Step 1: Story Generator

The goal is to create a program that generates the story (two statements and a question). The first sentence will be composed of a random selection of words.

MIMI HAD 10 CAKES

Let's work on these parts of the sentence: the noun, the number, and the object.

We'll make a list of nouns and a list of objects from which to choose. Using the command MAKE, we'll name them NOUNS and OBJECTS.

?MAKE "NOUNS [MIMI SUE JESSIE KARIN]

?MAKE "OBJECTS [CAKES TOYS BOOKS PENS]

We need a way to select a word randomly from each of these two lists. This can be done by a general purpose procedure that receives a list as input and returns one of the elements in this list.

```
TO PICK :LIST
  OUTPUT ITEM (1 + RANDOM COUNT :LIST) :L →
  IST
END
```

PICK is an operation. It outputs an element of the input list extracted by ITEM. ITEM takes two inputs. The first one, 1 + RANDOM COUNT :LIST, specifies the position of the element. The second, :LIST, specifies which list to choose from.

Let's try PICK with NOUNS as its input.

```
?PRINT PICK :NOUNS
SUE
?PRINT PICK :NOUNS
JESSIE
```

We can now generate the first sentence. PICK :NOUNS will output the name. PICK :OBJECTS will output an object. We'll use RANDOM 10 to get a random number. The primitive SENTENCE (SE) will combine all of the elements into a sentence.

```
?PR (SE PICK :NOUNS "HAD RANDOM 10 PICK →
:OBJECTS)
```

We can now write a procedure to generate the whole story.

```
TO STORY
  PR (SE PICK :NOUNS "HAD RANDOM 10 PICK →
  :OBJECTS)
  PR SE [SHE BOUGHT] RANDOM 10
  PR [HOW MANY DID SHE HAVE THEN?]
END
```


Try STORY.

?STORY

The result should be similar to:

JESSIE HAD 2 TOYS
SHE BOUGHT 9
HOW MANY DID SHE HAVE THEN?

?STORY
MIMI HAD 4 CAKES
SHE BOUGHT 1
HOW MANY DID SHE HAVE THEN?

Step 2: An Arithmetic Problem

The next step is to turn our story generator into an arithmetic problem generator. The program must compute the solution (using the two numbers given in the program) and evaluate the answer as well. In this version, the answer can always be found by addition.

First, we have to remember the two numbers in the problem as well as their sum.

MAKE "NUM1 RANDOM 10

:NUM1 is the first number appearing in the problem.

MAKE "NUM2 RANDOM :NUM1

:NUM2 is the second number and appears in the second sentence. Note that RANDOM :NUM1 will always output a random number less than whatever NUM1 is. Therefore, when we use subtraction, we won't have to deal with negative numbers.

MAKE "ANS :NUM1 + :NUM2

:ANS is the sum of both numbers.

We'll write a new procedure, **SETUP**, to set up all of the variables.

```
TO SETUP
MAKE "NOUNS [MIMI SUE JESSIE KARIN]
MAKE "OBJECTS [CAKES TOYS BOOKS PENS]
MAKE "NUM1 RANDOM 10
MAKE "NUM2 RANDOM :NUM1
MAKE "ANS :NUM1 + :NUM2
END
```

Edit **STORY** to use **:NUM1** and **:NUM2** instead of **RANDOM 10**.

```
TO STORY
PR (SE PICK :NOUNS "HAD :NUM1 PICK :OBJ→
ECTS)
PR SE [SHE BOUGHT] :NUM2
PR [HOW MANY DID SHE HAVE THEN?]
END
```

The superprocedure, **PROBGEN**, first will run **SETUP** and **STORY**, then pick up the answer with the operation **READWORD**, verify whether it's right, and respond accordingly. We'll make **PROBGEN** recursive, so it will run a new problem when the first problem is finished.

```
TO PROBGEN
SETUP
STORY
TEST :ANS = READWORD
IFTRUE [PR [RIGHT!]]
IFFALSE [PR SE [GOOD TRY BUT THE ANSWER→
IS] :ANS]
PROBGEN
END
```

Let's try PROBGEN.

?PROBGEN

JESSIE HAD 7 PENS

SHE BOUGHT 3

HOW MANY DID SHE HAVE THEN?

10

RIGHT!

SUE HAD 5 BOOKS

SHE BOUGHT 1

HOW MANY DID SHE HAVE THEN?

4

GOOD TRY BUT THE ANSWER IS 6

Press Ctrl-Break to stop.

Step 3: Variation in Problems

PROBGEN works fine but lacks variety because the examples only have been addition problems with female names.

To vary the arithmetic operation, we must change the verb in the second sentence of the problem. We'll use only addition and subtraction verbs to keep things simple. BOUGHT and FOUND are addition verbs. SOLD and LOST are subtraction verbs. We use a list of pairs to tell Logo which verbs are associated with which arithmetic operation.

```
?MAKE "VERBOP [[BOUGHT +] [FOUND +] [SO→  
LD -] [LOST -]]
```

This is a list of lists. Each list consists of a verb and its associated operation. VERBOP is the name of the complete list.

How can we access the information contained in this list? We used PICK to output a random word in the list it had been given as input. PICK can be used in the same way for VERBOP so it outputs a list of two elements.

```
?PRINT PICK :VERBOP  
BOUGHT +
```

Use the primitive FIRST to pick out the verb.

```
?PRINT FIRST PICK :VERBOP  
SOLD
```

To pick out the arithmetic operation, use the primitive LAST.

```
?PRINT LAST PICK :VERBOP  
+
```

Each time we use PICK it outputs an element of VERBOP. This element is different every time because we want the verb, FIRST PICK :VERBOP, to correspond to the operation, LAST PICK :VERBOP. Thus,

```
MAKE "VERBOP PICK [[BOUGHT +] [FOUND +]→  
[SOLD -] [LOST -]]
```

Next create a list of pairs for the male and female names. Each name is associated with a pronoun (HE SHE).

```
MAKE "NOUNPRO PICK [[SUE SHE] [BRUCE HE→  
] [MIMI SHE] [JOE HE]]
```

The first sentence in our problem uses FIRST :NOUNPRO (the name). The second and third sentences use LAST :NOUNPRO (the pronoun).

SETUP should look like this:

```
TO SETUP
MAKE "NOUNPRO PICK [[SUE SHE] [BRUCE HE→
] [MIMI SHE] [JOE HE]]
MAKE "OBJECT PICK [CAKES BOOKS COMPUTER→
S GAMES DOGS]
MAKE "VERBOP PICK [[BOUGHT +] [FOUND +]→
[SOLD -] [LOST -]
MAKE "NUM1 RANDOM 10
MAKE "NUM2 RANDOM :NUM1
MAKE "ANS RUN (SE :NUM1 LAST :VERBOP :N→
UM2)
END
```

Notice the change in the last line of SETUP. RUN runs the combination of LAST:VERBOP (the arithmetic operation) and the variables :NUM1 and :NUM2 to compute the result.

Edit STORY to specify the new variables.

```
TO STORY
PR (SE FIRST :NOUNPRO "HAD :NUM1 :OBJEC→
T)
PR (SE LAST :NOUNPRO FIRST :VERBOP :NUM→
2)
PR (SE [HOW MANY DID] LAST :NOUNPRO [HA→
VE THEN?])
END
```

Try the superprocedure PROBGEN again. The result should look similar to our opening scenario.

Suggestions for Modifications

There are many ways to expand the PROBGEN program. Here are some suggestions:

- Make PROBGEN stop. This can be done by pressing any key, such as Q. Another possibility is to stop PROBGEN when it has run a certain number of times.
- Keep score of the amount of “RIGHTS” and “WRONGS”. Get PROBGEN to give you the score after each problem is completed, or just before you decide to quit.
- Add new nouns and verbs to the existing lists. This could be done by writing a procedure that asks for either a new “given name” and its corresponding pronoun or a new verb and its corresponding arithmetic operation.
- Extend the arithmetic operations to include multiplication and division.
- Ask for a maximum number to be used in the arithmetic problems. The maximum number will be in the input to RANDOM in giving NUM1 a value.
- Use graphics (if available) to animate the program. A simple possibility is to draw a happy face for a right answer and a sad face for a wrong answer.

Program Listing

```
TO SETUP
MAKE "NOUNPRO PICK [[SUE SHE] [BRUCE HE→
] [MIMI SHE] [JOE HE]]
MAKE "OBJECT PICK [CAKES BOOKS COMPUTER→
S GAMES DOGS]
MAKE "VERBOP PICK [[BOUGHT +] [FOUND +]→
[SOLD -] [LOST -]]
MAKE "NUM1 RANDOM 10
MAKE "NUM2 RANDOM :NUM1
MAKE "ANS RUN (SE :NUM1 LAST :VERBOP :N→
UM2)
END
```

```
TO STORY
PR (SE FIRST :NOUNPRO "HAD :NUM1 :OBJEC→
T)
PR (SE LAST :NOUNPRO FIRST :VERBOP :NUM→
2)
PR (SE [HOW MANY DID] LAST :NOUNPRO [HA→
VE THEN?])
END
```

```
TO PICK :LIST
OUTPUT ITEM (1 + RANDOM COUNT :LIST) :L→
IST
END
```

```
TO PROBGEN
SETUP
STORY
TEST :ANS = READWORD
IFTRUE [PR [RIGHT!]]
IFFALSE [PR SE [GOOD TRY BUT THE ANSWER→
IS] :ANS]
PROBGEN
END
```

Part 2. Reference Information

Chapter 7. Logo Primitives

Contents

Introduction	7-3
Special Keys	7-9
Special Words	7-12
Infix Operations	7-13
.(Dot) Primitives	7-13
Alphabetical Listing of Primitives	7-13

Introduction

This chapter consists of a description of each primitive of IBM Logo. The descriptions are in the following order:

- Special keys
- Special words
- Operations whose names are symbols rather than words +, -, /, *, =, >, <
- Primitives whose names begin with a period (.)
- The alphabetical listing

At the top of each description, you will find:

- The name of the primitive.
- The short form of the name, if one exists. You will see this in parentheses after the long form.
- Whether the primitive is a command, an operation or an infix operation. The difference between a command and an operation is described in Chapter 2, “Logo Grammar.” An infix operation has its symbol placed between the inputs. All other primitives are written in front of their inputs.

The description itself has three sections:

Format: The name of the primitive, followed by the type of each input. All primitives *must* be entered in uppercase letters. You are to supply all inputs (shown in *italics*).

Remarks: General information about the primitive.

Examples: Illustrations of how to use the primitive.

In some cases, there is a fourth section called "Comments," which contains supplementary technical information.

If a primitive has more than one format, the simplest or most commonly used one is presented first. You will see that, with some primitives (such as SUM), an optional format is enclosed in parentheses. This indicates that the primitive will accept as many inputs as you wish. When using more than two inputs with such a primitive (or, in some cases, one input) you must always put a left parenthesis before its name and a right parenthesis after the last input.

When we describe the kind of input that a primitive requires, we are *not* speaking about the way the input is written when you define the procedure, the rules for which were described in Chapter 2, "Logo Grammar." When Logo tries to understand a written input, it evaluates it, thereby changing it to something else. The following chart shows what these changes are. For example, if we write

MAKE :X 22 + 23

and X contains the word JOHN, then the real inputs to MAKE are the word JOHN and the number 45.

Written input	Real input
Word with quote in front	Word
Word with colon in front	Contents of word. This can be a word, list or number.
Number	Number
List	List
Procedure with inputs	Output of procedure. This can be a word, list or number.

In this section and throughout Chapter 7, when we describe the kind of input that a primitive requires, we are speaking about the *real* input. With many primitives, an input can be anything. In other words, the real input can be a word, list, or number. We call this kind of input a Logo object. If we look up **MAKE**, you will see that it must have the following form:

MAKE *name object*

MAKE uses two input words: *name* and *object*. The input *name* means that the first input must be a word (we call a word a name if it is to be the name of something like a variable or a procedure). The input *object* is an abbreviation for a Logo object. Going back to our example, we see that **JOHN** is a word and **45** is a Logo object, so we do have the correct inputs.

All of the input words and abbreviations that we use in describing the inputs to the Logo primitives are explained on the next few pages.

Input Words	Meaning
<i>character (char)</i>	A character. See Chapter 2, “Logo Grammar.”
<i>colorlist</i>	A list of two numbers, foreground and background colors. See the SETTC command.
<i>colornumber</i>	An integer, from 0 through 3 for pen color, from 0 through 15 for background.
<i>degrees</i>	An angle in degrees (not radians); real number.
<i>device</i>	A device name. See “Device Names” in Chapter 4.
<i>distance</i>	A distance expressed in turtle steps: a real number from -9999 to 9999.
<i>drive</i>	A drive name: A, B, C, etc.
<i>duration (dur)</i>	Time specified in ticks: a number from 0 to 9999.
<i>ext</i>	An extension of a file name. See “Naming Files” in Chapter 4.
<i>file name</i>	A file name. See “Naming Files” in Chapter 4.
<i>filepos</i>	The position of the file in bytes: a number from 0 to the total length of the file.
<i>filespec</i>	File specification. See “Naming Files” in Chapter 4.

Input Words	Meaning
<i>freq</i>	A frequency specified in Hertz: a real number from 37 to 19723.
<i>infilespec</i>	A file name.
<i>input</i>	Word with colon preceding it. Used only with TO.
<i>instructionlist</i>	List of procedures that Logo can execute.
<i>length</i>	Length of a file in bytes.
<i>list</i>	Information enclosed in [] brackets.
<i>n, a, b, x, y</i>	A number.
<i>name</i>	A word naming a procedure or a variable.
<i>namelist</i>	A list of names.
<i>newname</i>	A word naming a new procedure or variable.
<i>object (obj)</i>	A Logo object (a word, list, or number).
<i>outfilespec</i>	A new file name.
<i>package</i>	The name of a collection of procedures and variables. See "Organizing Your Procedures in Files" in Chapter 4.
<i>packagelist</i>	A list of packages.
<i>paddlenumber</i>	An integer from 0 to 3.

Input Words	Meaning
<i>penlist</i>	A list of three elements. See the SETPEN command.
<i>position (pos)</i>	A list of two numbers: x- and y- coordinates, or cursor location.
<i>pred</i>	A predicate—an operation that outputs either the word TRUE or the word FALSE.
<i>prop</i>	A property. See the GPROP operation and the PPROP command.
<i>word</i>	A literal word, or sequence of characters considered as one unit. See Chapter 2, “Logo Grammar.”

The following input words are specific to the assembly language primitives. For their possible values, see the primitive definitions and Appendix C, “How Logo Uses Memory.”

base
bytevalue
offset

The .SETCOM command has specific input words. Their values are given in the definition of .SETCOM.

bdr
parity
databits
stopbits

Special Keys



Backslash

*Quotes next character you type, allowing you to quote delimiters such as a space.



Backspace

*Erases character to left of cursor.



Cursor Left

*Moves cursor one space to the left.



Cursor Right

*Moves cursor one space to the right.




Shift



*  Moves cursor to beginning of current line.



*  Moves cursor to end of current line.



Cursor Up

Moves cursor up to previous line in editor.



Cursor Down

Moves cursor down to next line in editor.



Enter

*Carriage return.



*Erases everything on current line to right of cursor.



*Inserts last line deleted.














Moves cursor to end of the edit buffer.



Moves cursor to top of the edit buffer.

*Indicates a key function which works both inside and outside of the editor.

Ctrl-Break	Exit from the editor without saving changes. Outside editor, interrupts running procedure, by stopping it.
 	Interrupts whatever is running. Typing any key resumes running.
	*Deletes character at cursor position.
	Moves cursor to end of current page in editor.
	Exits from editor, reading buffer as if typed in.
	Outside editor, corresponds to TEXTSCREEN.
	Outside editor, corresponds to MIXEDSCREEN. If F2 does not have any effect, use the MS command instead. See the CLEARTEXT command for details.
	*Inserts a copy of the last line typed.
	Outside editor, corresponds to FULLSCREEN. If F4 does not have any effect, use the FS command instead. See the CLEARTEXT command for details.
	Outside editor, corresponds to PAUSE.
	Moves cursor to top of current page in editor.

*Indicates a key function which works both inside and outside of the editor.



Inserts new line at the position of the cursor in editor.



*Sets the numeric keypad to work more like a calculator keypad.



Scrolls screen to next page of editor.



Scrolls screen to previous page of editor.



Shift



*Prints the screen onto the printer.
Will only print graphics if a graphics printer is attached.

*Indicates a key function which works both inside and outside of the editor.

Special Words

EDITOR	Input for EDIT or EDITFILE, standing for contents of the edit buffer.
END	Tells Logo that you are finished defining a procedure. See the TO command.
ERRACT	System variable: if TRUE, Logo pauses when error occurs. See the ERROR operation.
ERROR	Tag for THROW when error occurs. See the CATCH command and the ERROR operation.
FALSE	Special input for AND, IF, NOT, OR, SETCAPS, and TEST.
PROCPKG	Property of procedure name, its value is the package name. See PACKAGE, PKGALL, and PPROP for giving the PROCPKG property to a procedure name.
REDEFP	System variable: if TRUE, primitives can be redefined. See the COPYDEF command.
STARTUP	System variable: if a list, Logo runs it immediately after the file containing this variable is loaded. See Appendix E.

.SYSTEM	Package containing ERRACT and REDEFP (initially buried). See the BURY command.
TOPLEVEL	Tag for THROW to return control to top level.
TRUE	Special input for AND, IF, NOT, OR, SETCAPS, and TEST.
TURTLE	Input for SETSHAPE, standing for normal turtle shape. Also can be output of SHAPE.
VALPKG	Property of variable name, its value is the package name. See PKGALL and PPROP for giving the VALPKG property to a variable.

Infix Operations

Refer to pages 7-14 through 7-21.

.(Dot) Primitives

Refer to pages 7-24 through 7-38.

Alphabetical Listing of Primitives

Refer to pages 7-39 through 7-352.

+ **Infix Operation**

Format: $a + b$

Remarks: Outputs the sum of its inputs, a and b . The $+$ operation may also be used as a prefix operation like SUM. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 5 + 2
 7

 ?PRINT 2.54 + 12.3 + 1.11
 15.95

Note that if you type

5 + 3

without PRINT, Logo prints the error message:

I DON'T KNOW WHAT TO DO WITH 8

The following procedure counts without stopping:

```
TO COUNTUP :N
PR :N
COUNTUP :N + 1
END
```

```
?COUNTUP 1
1
2
3
4
5
. . .
```

Infix Operation

Format: $a - b$

Remarks: Outputs the result of subtracting b from a . The $-$ operation may be used in several ways:

- an infix operation with two inputs

```
?PRINT 4 - 3  
1
```

- to indicate a negative number. In this case, there should be no space between the minus sign and the number.

```
?PRINT -3  
-3
```

In addition the $-$ operation can be used as a prefix operation (like DIFFERENCE) with two inputs. See “Special Prefix Operations” in Chapter 5.

Examples:

```
?PRINT -XCOR  
-50
```

```
?PRINT - 3  
NOT ENOUGH INPUTS TO -
```


Infix Operation

Note that there could be a confusion between the minus sign with one input and the minus sign with two inputs. Logo resolves this as follows:

```
?PRINT 7 - 1      (7 minus 1)
6
```

```
?PRINT 7-1        (7 minus 1)
6
```

```
?PRINT 7 - -1     (7 minus negative 1)
8
```

```
?PRINT PRODUCT 7 -1 (7 times negative 1)
-7
```

```
?PRINT 3 * -4     (3 times negative 4)
-12
```

```
?PRINT (3+4)-5    ((3 plus 4) minus 5)
2
```

The procedure ABS outputs the absolute value of its input.

```
TO ABS :NUM
  OUTPUT IF :NUM < 0 [-:NUM] [:NUM]
END
```

```
?PRINT ABS -35
35
```

```
?PRINT ABS 35
35
```

Infix Operation

The procedure NEAR tells whether two numbers are “close” to each other.

```
TO NEAR :A :B  
  OUTPUT (ABS :A - :B) < (:A / 1000)  
END
```

```
?PRINT NEAR XCOR 100  
TRUE
```

```
?PRINT XCOR  
99.993
```

*

Infix Operation

Format: $a * b$

Remarks: Outputs the product of its inputs a and b . The $*$ operation may also be used as a prefix operation like **PRODUCT**. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 2 * 3
6

?PRINT 2 + 3 * 4
14

Multiplication is done before addition. See “Order of Mathematical Operations” in Chapter 5.

?PRINT 1.3 * -1.3
-1.69

?PRINT 48 * (.3 + .2)
24

The procedure **FACTORIAL** outputs the factorial of its input. For example, **FACTORIAL 5** outputs the result of $5 * 4 * 3 * 2 * 1$.

```
TO FACTORIAL :N
IF :N = 0 [OUTPUT 1] [OUTPUT :N * FACTO→
RIAL :N-1]
END
```

?PRINT FACTORIAL 4
24

?PRINT FACTORIAL 1
1

/

Infix Operation

Format: a / b

Remarks: Outputs the result of a divided by b . The $/$ operation also may be used as a prefix operation like QUOTIENT. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 6 / 3
2

?PRINT 8 / 3
2.666666667

?PRINT 2.5 / -3.8
-0.6578947368

?PRINT 0 / 7
0

?PRINT 7 / 0
CAN'T DIVIDE BY ZERO

The POLYGON procedure draws a regular polygon with the angle specified by the input for the corners. It calculates the number of sides by dividing 360 by :ANGLE.

```
TO POLYGON :ANGLE
REPEAT 360 / :ANGLE [FD 20 RT :ANGLE]
END
```

<

Infix Operation

Format: $a < b$

Remarks: Outputs TRUE if a is less than b ; otherwise, outputs FALSE. The $<$ operation may also be used as a prefix operation. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 2 < 3
 TRUE

 ?PRINT -7 < -10
 FALSE

The procedure BETWEEN outputs TRUE if the number given as the first input is greater than the second and less than the third.

TO BETWEEN :NUM :LOW :HI
OUTPUT AND :LOW < :NUM :NUM < :HI
END

?PRINT BETWEEN 5 2 6
TRUE

?PRINT BETWEEN 7 2 6
FALSE

> Infix Operation

Format: $a > b$

Remarks: Outputs TRUE if a is greater than b ; otherwise, outputs FALSE. The $>$ operation also may be used as a prefix operation. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 4 > 3
 TRUE

 ?PRINT -10 > -7
 FALSE

The procedure OUTSIDE outputs TRUE if the number given as the first input is less than the second input or greater than the third.

```
TO OUTSIDE :N :LOW :HI
OP OR :LOW > :N :N > :HI
END
```

```
?PRINT OUTSIDE 15 0 16
FALSE
```

```
?PRINT OUTSIDE -5 -2 5
TRUE
```


=

Infix Operation

Format: *object1* = *object2*

Remarks: Outputs TRUE if *object1* and *object2* are equal numbers, identical words, or identical lists; otherwise, outputs FALSE. The = operation may also be used as a prefix operation like EQUALP. See “Special Prefix Operations” in Chapter 5.

Examples: ?PRINT 100 = 50 * 2
TRUE

?PRINT 3 = FIRST "3.1416
TRUE

?PRINT [THE A AN] = [THE A]
FALSE

?PRINT 7. = 7
TRUE

(A decimal number is equivalent to the corresponding integer.)

?PRINT " = []
FALSE

(The empty word and the empty list are not identical.)

The procedure FLIP prints HEADS if the result of the RANDOM toss is 1. Otherwise, it prints TAILS.

```
TO FLIP
IF 1 = RANDOM 2 [PRINT [HEADS]] [PRINT →
[TAILS]]
END
```

.BLOAD Command

Format: .BLOAD *filespec*
 (.BLOAD *filespec base offset*)

Remarks: Stands for “Binary LOAD”.

Loads a binary-format file (assembly language or data) into memory.

If the file was saved by Logo or BASIC, the first format may be used. In this case, the file will be loaded wherever it resided in memory when saved.

If the file is to be placed in a different area of memory, the second format must be used.

For the possible values of *base* and *offset*, see “Reserving High End Memory Space” and “Values of *base* and *offset*” in Appendix C. More detailed information can be found in the *IBM Personal Computer MACRO Assembler* manual.

The second format must also be used if the file does not have the standard file header created by Logo or BASIC. The header created by .BSAVE and used by .BLOAD is 7 bytes long and is identified by its first byte, which always contains “FD”.

.BLOAD Command

Although .EXE files also have headers, they won't be recognized and .BLOAD will load them as data. To load .EXE files, see "Assembly Language Subroutines" in Appendix C.

Warning: Primitives with leading periods require some understanding of the machine to be used effectively and safely. Although learning by experimentation is encouraged, a careless .DEPOSIT or .CALL, for instance, may destroy your workspace.

Examples: `?(.BLOAD "CHARSET3 16384 128)`

This statement could be used to load a character set from disk. Had it been saved from the specified *base* and *offset*, the following would do the same:

`?.BLOAD "CHARSET3`

.BSAVE Command

Format: *.BSAVE filespec base offset length*

Remarks: Stands for “Binary SAVE”.

Copies an area of computer memory to a file whose name is the first input. For information on naming files, see Chapter 4. The memory area transferred begins at the address specified by *base* and *offset*.

For the possible values of *base* and *offset*, see “Reserving High End Memory Space” and “Values of *base* and *offset*” in Appendix C. The *length* (in bytes) of the area transferred is specified by the fourth input. The *length* cannot be greater than 65535.

Logo, like BASIC, adds a seven-byte header at the beginning of the file. The first byte of this header is hexadecimal “FD”. More detailed information can be found in the *IBM Personal Computer MACRO Assembler* manual.

Example: ?*.BSAVE "FASTSCRN 6144 2048 78*

This statement would save 78 bytes in a file named FASTSCRN.

.CALL

Command

Format: *.CALL base offset*

Remarks: Logo passes control to the indicated assembly language subroutine, starting at the address specified by *base* and *offset*.

For possible values of *base* and *offset*, see “Reserving High End Memory Space” and “Values of *base* and *offset*” in Appendix C.

The *.CALL* command saves the four registers Logo uses: DS, ES, DX, BP and the flags.

The *.CALL* command does a FAR call so the return instruction should be coded accordingly.

More detailed information can be found in the *IBM Personal Computer MACRO Assembler* manual.

See *.DEPOSIT* for an example of the use of *.CALL*.

Warning: Primitives with leading periods require some understanding of the machine to be used effectively and safely. Although learning by experimentation is encouraged, a careless *.DEPOSIT* or *.CALL*, for instance, may destroy your workspace.

.CONTENTS

Operation

Format: .CONTENTS

Remarks: Outputs a list of all objects that Logo knows about. This includes your variables and procedures, the Logo primitives, most of what you've typed, and some other words.

Note that the last object in .CONTENTS is an empty word (a space).

Examples: ?PRINT LAST .CONTENTS

```
?PRINT LAST BUTLAST .CONTENTS  
WORD
```


.DEPOSIT

Command

Format: *.DEPOSIT base offset bytevalue*

Remarks: Stores the value indicated by *bytevalue* at the address specified by *base* and *offset*.

For the possible values of *base* and *offset*, see “Reserving High End Memory Space” and “Values of *base* and *offset*” in Appendix C.

Bytevalue must be from 0 through 255; again inputs are in decimal. See the .EXAMINE operation.

More detailed information can be found in the *IBM Personal Computer MACRO Assembler* manual.

Warning: Primitives with leading periods require some understanding of the machine to be used effectively and safely. Although learning by experimentation is encouraged, a careless .DEPOSIT or .CALL, for instance, may destroy your workspace.

Example: TO DEPO :BASE :OFFSET :BYTES
IF EMPTY :BYTES [STOP]
.DEPOSIT :BASE :OFFSET FIRST :BYTES
DEPO :BASE :OFFSET + 1 BF :BYTES
END

?SAVE "DEPO

? .DOS

A>LOGO 1

?LOAD "DEPO

?DEPO 8000 0 [184 10 7 185 20 10 186 30→
20 183 240 205 16 203]

DEPO stores a list of byte values at the address specified by its inputs. In this case the list of byte values is a machine language program that you can execute.

? .CALL 8000 0

A block appears on the screen.

.DOS

Command

Format: .DOS

Remarks: Logo releases control to the Disk Operating System. All the information in your workspace is lost. To return to Logo, you must respond to the DOS prompt by typing

A>LOGO

Your Logo Language Diskette contains DOS 2.00 and the following DOS 2.00 commands:

DISKCOPY to make duplicate copies of diskettes.

FORMAT to format diskettes.

GRAPHICS to print graphics on the IBM Graphics Printer. (Is loaded by AUTOEXEC.BAT. See “Starting Logo” in Chapter 1.)

For more information about these programs, see your *IBM Personal Computer DOS 2.00 Reference* manual.

.EXAMINE

Operation

Format: *.EXAMINE base offset*

Remarks: Outputs the byte stored at the address indicated by *base* and *offset*.

For the possible values of *base* and *offset*, see “Reserving High End Memory Space” and “Values of *base* and *offset*” in Appendix C.

The output will be an integer from 0 through 255. See the *.DEPOSIT* command.

.EXAMINE and *.DEPOSIT* are complementary. They are useful for data storage and passing information to and from machine-code routines and auxiliary memory-mapped hardware such as real-time clocks.

More detailed information can be found in the *IBM Personal Computer MACRO Assembler* manual.

.EXAMINE

Operation

Example: `TO STARTTIMER`
`MAKE "START.TIME READTIME`
`END`

`TO READTIME`
`OP (.EXAMINE 64 110) * 65536 + (.EXAMIN→`
`E 64 109) * 256 + .EXAMINE 64 108`
`END`

`TO ELAPSEDTIME`
`LOCAL "RESULT`
`MAKE "RESULT READTIME - :START.TIME`
`IF :RESULT < 0 [MAKE "RESULT 1573040 + →`
`:RESULT]`
`OP INT :RESULT / 18.2`
`END`

This set of procedures allows you to access the time which is kept by the BIOS.

The BIOS (Basic Input Output System) is stored on Read-Only-Memory (ROM) and is a part of DOS.

STARTTIMER sets the time to the time kept by the BIOS. ELAPSEDTIME outputs the elapsed time in seconds from the time that you set STARTTIMER to the time that you execute ELAPSEDTIME.

The number 1573040 is the number of ticks in 24 hours.

.SCREEN Operation

Format: .SCREEN

Remarks: Outputs 1 or 2:

- 1 for one screen and
- 2 for two screens (one IBM Monochrome Display and one monitor or TV).

The default value is 1.

To use the MIXEDSCREEN (MS), FULLSCREEN (FS), TEXTSCREEN (TS), and SETTEXT commands with an IBM Color/Graphics Monitor Adapter, .SCREEN must be 1. With an IBM Monochrome Display, MS, FS, and SETTEXT give the error message:

CAN'T DO GRAPHICS IN THIS MODE

See the .SETSCREEN command to change the value of .SCREEN.

Examples: ?PRINT .SCREEN

1

? .SETSCREEN 2

?PRINT .SCREEN

2

.SCRUNCH

Operation

Format: `.SCRUNCH`

Remarks: Outputs the aspect ratio, a decimal number which is the ratio of the size of a vertical turtle step (on the screen) to a horizontal one.

The aspect ratio is .8 when Logo starts up. This means that there are 320 dots available on the x-axis and 250 dots available on the y-axis.

The aspect ratio affects the screen coordinates in the POS operation. See the `.SETSCRUNCH` command.

.SETCOM Command

Format: `.SETCOM n bdrp parity databits stopbits`

Remarks: The .SETCOM command takes five inputs and uses them to set up a serial communications line. You must have an Asynchronous Communications Adapter in order to use this command. If not, Logo will print the following message:

DEVICE UNAVAILABLE

You must use .SETCOM if you are sending information on a serial line to a device such as a serial printer. The following values are accepted when ordered correctly:

n (Asynchronous Communications Adapter number): May be 1 or 2.

bdrp (baud rate, transmission speed): May be any of 110, 150, 300, 600, 1200, 2400, 4800, or 9600 baud.

parity: May be 0 (none).
1 (odd).
2 (even).

databits: May be 7 or 8.

stopbits: May be 1 or 2.

To set up a serial printer, you must use .SETCOM with the appropriate values for your particular printer. See "Printing Files" in Chapter 4 for further information.

.SETSCREEN

Command

Format: `.SETSCREEN n`

Remarks: The *n* must be 1 or 2.

If *n* is 1, `.SETSCREEN` sets the output to one screen.

If *n* is 2, `.SETSCREEN` sets the display to two screens: one IBM Monochrome Display for text and one monitor or TV for graphics. If two screens are not connected, Logo will print the following error message:

`DEVICE UNAVAILABLE`

When you use `.SETSCREEN 2`, the `MIXEDSCREEN`, `FULLSCREEN`, `TEXTSCREEN`, and `SETTEXT` primitives have no effect. In addition, the function keys F1, F2, and F4, have no effect at top level. See the `.SCREEN` operation.

Examples: `? .SETSCREEN 2`

sets the display to two screens: one for graphics, one for text.

`? .SETSCREEN 1`

sets the display back to one screen.

.SETSCRUNCH Command

Format: .SETSCRUNCH n

Remarks: Sets the aspect ratio (see the .SCRUNCH command) to n , where n can be any number from 0 to 9999. YCOR is changed accordingly.

Examples: .SETSCRUNCH .5 makes each vertical turtle step half the length of a horizontal one.

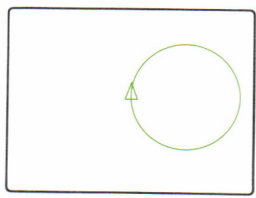
.SETSCRUNCH has two uses. It is primarily intended to be used when “squares” turn out looking like rectangles and “circles” turn out looking like ellipses on some screens. (.SCRUNCH=.8 is the aspect ratio Logo begins with and is correct for most screens.)

A second use for .SETSCRUNCH is to transform turtle drawings by squashing or extending them. This can add a great deal of power to simple graphics procedures, as in the example below, which creates circles and ellipses with a given radius.

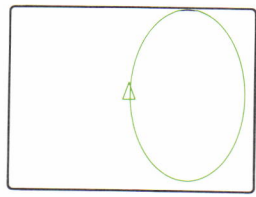
```
TO CIRCLE :RADIUS
REPEAT 60 [FD :RADIUS * 3.14159 / 30 RT→
6]
END
```

```
TO ELLIPSE :HORIZ :VERT
MAKE "OLDASPECT .SCRUNCH
.SETSCRUNCH .8 * :VERT / :HORIZ
CIRCLE :HORIZ
.SETSCRUNCH :OLDASPECT
END
```

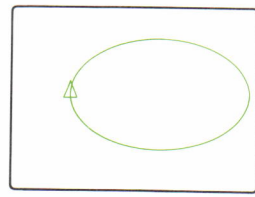
.SETSCRUNCH Command



?CS
?CIRCLE 5



?CS
ELLIPSE 5 8



?CS
ELLIPSE 8 5

ALLOPEN

Operation

Format: ALLOPEN

Remarks: Outputs a list of all files and devices currently open.
See the OPEN command to open a file or device.

Examples: ?PRINT ALLOPEN

?

No files or devices are open.

```
?OPEN "LPT1
?OPEN "BOOKLIST
?PRINT ALLOPEN
LPT1 BOOKLIST
```

The printer and the file called BOOKLIST are open.

The procedure BYE makes sure all files are closed
before you turn off the machine.

```
TO BYE
IF NOT EMPTY ALLOPEN [CLOSEALL]
PR [YOU CAN NOW TURN OFF THE POWER.]
END
```


AND

Operation

Format: *AND pred1 pred2*
 (*AND pred1 pred2 pred3 . . .*)
 (*AND pred*)

Remarks: Receives one or more inputs. Its inputs must be TRUE or FALSE. AND outputs TRUE if all of its inputs are true; otherwise FALSE.

Examples: ?PRINT AND "TRUE "TRUE
 TRUE

 ?PRINT AND "TRUE "FALSE
 FALSE

 ?PRINT AND "FALSE "FALSE
 FALSE

 ?PRINT (AND "TRUE "TRUE "FALSE "TRUE)
 FALSE

 ?PRINT AND 5 7
 5 IS NOT TRUE OR FALSE

 ?PRINT AND PENCOLOR = 0 BACKGROUND = 0
 FALSE

(The infix operation = returns TRUE or FALSE to AND.)

AND Operation

The following procedure, DECIMALP, tells whether or not its input is a decimal number with a non-zero fractional part:

```
TO DECIMALP :OBJ
  OUTPUT AND NUMBERP :OBJ MEMBERP "." :OBJ
END
```

```
?PRINT DECIMALP 17
FALSE
```

```
?PRINT DECIMALP 17.
FALSE
```

(The decimal point is not kept internally.)

```
?PRINT DECIMALP 17.0
FALSE
```

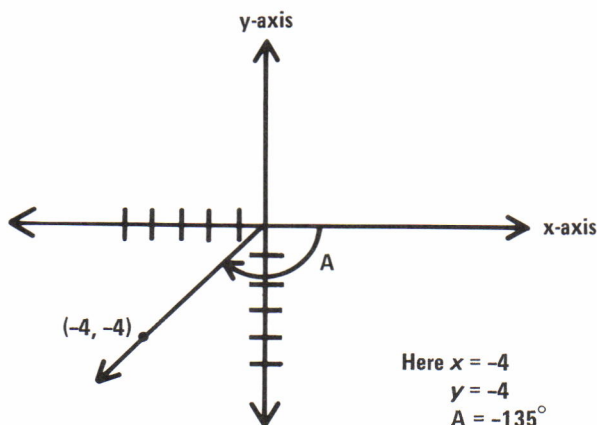
```
?PRINT DECIMALP 17.17
TRUE
```

ARCTAN

Operation

Format: $\text{ARCTAN } y$
 $\text{ARCTAN } y \ x$

Remarks: Outputs the angle, in degrees, between the positive x-axis of the Cartesian plane and the line segment joining the origin and the point (x, y) of the plane. The output will always be a number A in the range $-180 < A < 180$.



Using ARCTAN with one input is the same as $\text{ARCTAN } y \ 1$. $\text{ARCTAN } y$ outputs, in degrees, the angle whose tangent is the number y . In this case, the result will always be a number A in the range $-90 < A < 90$.

ARCTAN Operation

In all cases, the results are rounded to PRECISION significant digits, where PRECISION must be less than or equal to 100.

Notes:

1. The inputs x and y must not *both* be equal to 0.

```
?PRINT ARCTAN 0 0  
ARCTAN DOESN'T LIKE [0 0] AS INPUT
```

2. When x is positive then (ARCTAN :Y :X) gives the same result as (ARCTAN :Y/:X).
3. If PRECISION is greater than 100, Logo prints the error message:

```
ARCTAN DOESN'T LIKE PRECISION n
```

ARCTAN

Operation

Examples: ?SETPRECISION 5
?PRINT ARCTAN 3 3
45

?PRINT ARCTAN 3 2
56.31

?PRINT ARCTAN 3 -2
123.69

?PRINT ARCTAN -2 -5
-158.2

?PRINT ARCTAN 0 -4
180

?PRINT ARCTAN 3 0
90

?PRINT ARCTAN 2
63.435

?PRINT ARCTAN 1 SQRT 3
29.999

?PRINT (ARCTAN QUOTIENT -1 SQRT 3)
-29.999

?SETPRECISION 30
?MAKE "Y 16.421963541
?PRINT ARCTAN :Y
86.5153303814890055047747097579

?SETPRECISION 150
?PRINT ARCTAN :Y
ARCTAN DOESN'T LIKE PRECISION 150

ARCTAN Operation

These procedures define ARCSIN and ARCCOS:

```
TO ARCSIN :X
OUTPUT ARCTAN :X / (SQRT 1 - :X * :X)
END
```

```
TO ARCCOS :X
OUTPUT ARCTAN (SQRT 1 - :X * :X) / :X
END
```

This definition of ARCCOS will give the wrong result for negative values of :X. (The general formula involves imaginary numbers.)

```
?SETPRECISION 10
?PRINT ARCSIN 4 / 5
53.13010235
```

```
?PRINT ARCCOS 3 / 5
53.13010235
```

```
?PRINT ARCCOS 4 / 5
36.86989765
```

Notice that the input to ARCSIN and ARCCOS (which is a SIN value or COS value) cannot be greater than 1.

ARCTAN

Operation

Comments: You can numerically check the validity of certain mathematical identities such as

$$\arctan \left(\frac{\sin a}{\cos a} \right) = a \quad \text{for } -90 < a < 90$$

and

$$\frac{\sin (\arctan y)}{\cos (\arctan y)} = y \quad \text{for real } y.$$

This can be done by comparing for various choices of :A and :Y and various precisions

```
ARCTAN SIN :A COS :A with :A
```

and

```
(SIN (ARCTAN :Y)) / (COS (ARCTAN :Y))  
with :Y
```

Because these identities involve more than one operation in succession, roundoff errors may occur. Accordingly, there may be a slight difference in the last digits of the outputs when they are compared.

ASCII Operation

Format: ASCII *character*

Remarks: Outputs the ASCII code for *character*, a quoted character. Appendix D contains a chart of all ASCII codes. If the input word contains more than one character, ASCII uses only its first character. See the CHAR operation.

Examples: ?PRINT ASCII "B
66

?PRINT ASCII "CAT
67

Logo prints an error message when the input to ASCII is a list or an empty word.

?PRINT ASCII []
ASCII DOESN'T LIKE [] AS INPUT

ASCII Operation

In the next example, the procedure SECRETCODE makes a new word by using the Caesar cipher (adding 3 to each letter).

```
TO SECRETCODE :WD
IF EMPTY? :WD [OUTPUT " ]
OUTPUT WORD SECRETCODELET FIRST :WD SEC→
RETCODE BF :WD
END
```

```
TO SECRETCODELET :LET
MAKE "LETNUM (ASCII :LET) + 3
IF :LETNUM > ASCII "Z [MAKE "LETNUM :LE→
TNUM - 26]
OUTPUT CHAR :LETNUM
END
```

```
?PRINT SECRETCODE "CAT
FDW
```

```
?PRINT SECRETCODE "CRAYON
FUDBRQ
```

BACK (BK) Command

Format: BACK *distance*

Remarks: Moves the turtle *distance* steps back. Its heading does not change. The *distance* is any real number from -9999 through 9999.



CS



BACK 70

If you do not have an IBM Color/Graphics Monitor Adapter attached to your computer the following error message will be displayed:

CAN'T DO GRAPHICS IN THIS MODE

If you give a distance large enough to exceed the boundary of the screen, one of the following happens:

- In FENCE mode, the turtle does not move, and Logo prints the error message:

TURTLE OUT OF BOUNDS.

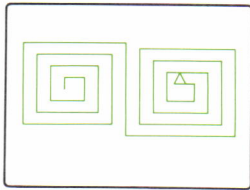
- In WINDOW mode, the turtle disappears off the edge of the screen.
- In WRAP mode, the turtle reappears from the opposite side of the screen.

BACK (BK)

Command

Examples: The following procedure draws a forward and backward turning spiral.

```
TO SURPRISE :STEP :INC  
IF :STEP > 80 [STOP]  
FORWARD :STEP  
RIGHT 90  
SURPRISE :STEP + :INC :INC  
BACK :STEP  
LEFT 90  
END
```



```
?SURPRISE 10 5
```

BACKGROUND (BG) Operation

Format: BACKGROUND

Remarks: Outputs a number representing the color of the current background for the graphics screen.

0 Black	8 Gray
1 Blue	9 Light Blue
2 Green	10 Light Green
3 Cyan	11 Light Cyan
4 Red	12 Light Red
5 Magenta	13 Light Magenta
6 Brown	14 Yellow
7 White	15 High-intensity White

You can think of colors 8 through 15 as “light” or “high-intensity” values of colors 0 through 7.

See the SETBG command to set the background colors.

Examples: ? SETBG 2

The background color of the graphics screen is set to 2 (Green).

```
?PRINT BACKGROUND  
2
```

BURY

Command

Format: BURY *package*

Remarks: Buries all procedures and variable names in *package*. See the PACKAGE command.

Buried procedures and names will not be included in the results of certain commands related to workspace (ERALL, ERNS, ERPS, POALL, PONS, POPS, POTS, and SAVE) unless the package name is specified as the input to these commands.

Buried variables or procedures can be erased if they are specified by name or package.

Each package name has a property called BURY. The BURY command makes the BURY property of the specified package TRUE, so that Logo will remember that this package is buried.

There is a predefined package called .SYSTEM that has the BURY property when Logo starts up. The .SYSTEM package contains the values for the special words ERRACT and REDEFP.

Examples: ?PACKAGE "USEFUL [ABS DIVISORP FOREVER →
POLY]
?BURY "USEFUL
?POTS
TO GREET
TO SETUP

?SAVE "FUNSTUFF saves the whole workspace in the file FUNSTUFF except for the procedures that were buried in the package USEFUL.

BURY Command

To see what is buried, you can use the command PPS:

```
?PPS  
USEFUL'S BURY IS TRUE  
POLY'S PROCPKG IS USEFUL  
FOREVER'S PROCPKG IS USEFUL  
DIVISORP'S PROCPKG IS USEFUL  
ABS'S PROCPKG IS USEFUL  
.SYSTEM'S BURY IS TRUE
```

In the following example, ABS would be erased even though it is in a buried package:

```
?ER "ABS
```

In the following example, all procedures in the buried package USEFUL would be erased.

```
?ERPS "USEFUL
```

BUTFIRST (BF)

Operation

Format: BUTFIRST *object*

Remarks: Outputs all but the first element of *object*.
BUTFIRST of the empty word or the empty list is an error.

Examples: ?SHOW BUTFIRST [IBM PERSONAL COMPUTER]
[PERSONAL COMPUTER]

?SHOW BUTFIRST "DOGS
OGS

BUTFIRST of a one-word list outputs an empty list.

?SHOW BUTFIRST [DOGS]
[]

?SHOW BUTFIRST [[THE A AN] [DOG CAT MOU→
SE] [BARKS MEOWS]]
[[DOG CAT MOUSE] [BARKS MEOWS]]

?SHOW BUTFIRST "
BUTFIRST DOESN'T LIKE AS INPUT

?SHOW BUTFIRST []
BUTFIRST DOESN'T LIKE [] AS INPUT

BUTFIRST (BF) Operation

The following procedure removes one element at a time from a word or a list.

```
TO TRIANGLE :MESSAGE  
IF EMPTY? :MESSAGE [STOP]  
PR :MESSAGE  
TRIANGLE BUTFIRST :MESSAGE  
END
```

```
?TRIANGLE "STROLL  
STROLL  
TROLL  
ROLL  
OLL  
LL  
L
```

```
?TRIANGLE [KANGAROOS JUMP GRACEFULLY]  
KANGAROOS JUMP GRACEFULLY  
JUMP GRACEFULLY  
GRACEFULLY
```

BUTLAST (BL)

Operation

Format: BUTLAST *object*

Remarks: Outputs all but the last element of *object*. BUTLAST of the empty word or the empty list is an error.

Examples: ?SHOW BUTLAST [I YOU HE SHE IT]
[I YOU HE SHE]

```
?SHOW BUTLAST "FLOWER  
FLOWE
```

```
?SHOW BUTLAST [FLOWER]  
[ ]
```

```
?SHOW BUTLAST "  
BUTLAST DOESN'T LIKE AS INPUT
```

```
?SHOW BUTLAST [ ]  
BUTLAST DOESN'T LIKE [ ] AS INPUT
```

```
?SHOW BUTLAST .123  
0.12
```

```
?SHOW BUTLAST ".123  
.12
```

The input to the following procedure should be an adjective ending in Y:

```
TO ONEUP :WD  
PR SE [YOU ARE] :WD  
PR SE [I AM] WORD BUTLAST :WD "IER  
END
```

```
?ONEUP "FUNNY  
YOU ARE FUNNY  
I AM FUNNIER
```

BUTTONP

Operation

Format: BUTTONP *paddlenumber*

Remarks: Outputs TRUE if the button on the specified paddle is down; otherwise FALSE. The *paddlenumber* must be 0, 1, 2, or 3 because there are only 4 paddles. BUTTONP 4 causes the error message:

BUTTONP DOESN'T LIKE 4 AS INPUT

When used with joysticks, stick 0 has button 0 and 1, and stick 1 has button 2 and 3. However, not all joysticks have two buttons.

If you do not have a Game Control Adapter, Logo prints the error message:

DEVICE UNAVAILABLE

If you have not plugged joysticks or paddles into the Game Control Adapter, BUTTONP 0, 1, 2, or 3 outputs FALSE.

Example: The procedure DRIVE allows you to control the turtle's movement with the button. The turtle will turn around to the right in a circle while you hold down the button of paddle number 0, to the left while you hold down button 1, and will go in a straight line when you release it.

```
TO DRIVE
IF BUTTONP 0 [RIGHT 5]
IF BUTTONP 1 [LEFT 5]
FORWARD 2
DRIVE
END
```

CAPS

Operation

Format: CAPS

Remarks: Outputs TRUE if the Caps Lock key is currently disabled. Outputs FALSE if it is active.

When you start Logo, CAPS is TRUE. This means everything that you type will be in uppercase, unless you hold the Shift key down.

You can change the value of CAPS with the SETCAPS command.

When CAPS is FALSE you can use the Caps Lock key like the Caps Lock key on a typewriter.

Example: TO CHECK
 TEST CAPS
 IFT [PR [CAPITAL LETTERS ONLY]]
 IFT [PR [UNLESS YOU PRESS SHIFT.]]
 IFF [PR [CAPS LOCK KEY IS ENABLED]]
 END

 ?CHECK
 CAPITAL LETTERS ONLY
 UNLESS YOU PRESS SHIFT.

 ?SETCAPS "FALSE
 ?CHECK
 CAPS LOCK KEY IS ENABLED

CAPS Operation

Press the Caps Lock key.

?now everything is in lowercase.
I DON'T KNOW HOW TO now

Press the Caps Lock key again.

?CHECK
CAPS LOCK KEY IS ENABLED

CATCH

Command

Format: *CATCH name instructionlist*

Remarks: Runs *instructionlist*. If a **THROW** with the same name is called while *instructionlist* is run, control returns to the **CATCH**. The *name* is used to match a **THROW** with a **CATCH**. For instance, **CATCH** “CHAIR *instructionlist* catches a **THROW** “CHAIR but not a **THROW** “TABLE.

There are two special cases. **CATCH** “TRUE catches any **THROW**; **CATCH** “ERROR catches an error that would otherwise print an error message and return to top level. If an error is caught, the message that Logo would normally print isn't printed. See the **ERROR** operation to identify the error.

Notes:

1. Ctrl-Break is treated as an error and would be caught by **CATCH** “ERROR.
2. **THROW** “TOPLEVEL, however, is not an error, so it is not caught by **CATCH** “ERROR.
3. **OUTPUT** and **STOP** do not behave like other primitives in a **CATCH**.

CATCH Command

Examples: The procedure SNAKE reads numbers that you type and uses them as distances to move the turtle. It turns the turtle between moves. If you type something other than a number, the program (using its READNUM subprocedure) prints an appropriate message and continues working.

```
TO SNAKE
CATCH "NOTNUM [SLITHER]
SNAKE
END
```

```
TO SLITHER
PRINT [TYPE A NUMBER, PLEASE.]
FORWARD READNUM READWORD
RIGHT 10
END
```

```
TO READNUM :NUM
IF NOT NUMBERP :NUM [PRINT [THAT'S NOT →
A NUMBER.] THROW "NOTNUM]
OUTPUT :NUM
END
```

To stop, press Ctrl-Break. (Notice that STOP would have returned to SLITHER, not to SNAKE.)

```
?SNAKE
TYPE A NUMBER, PLEASE.
4
TYPE A NUMBER, PLEASE.
25
TYPE A NUMBER, PLEASE.
T
THAT'S NOT A NUMBER.
TYPE A NUMBER, PLEASE.
```

CATCH

Command

In the next example, the procedure DOIT runs instructions that you type. When an error occurs, Logo does not issue the standard error message and does not return to top level; instead, it prints

```
THAT STATEMENT IS INCORRECT
```

and lets you continue to type instructions.

```
TO DOIT
CATCH "ERROR [DOIT1]
PRINT [THAT STATEMENT IS INCORRECT]
DOIT
END
```

```
TO DOIT1
RUN READLIST
DOIT1
END
```

```
?DOIT
PRINT 3 + 5
8
PRINT 12 - 7
THAT STATEMENT IS INCORRECT
PRINT 12 - 7
5
THROW "TOPLEVEL
?
```

Format: CHAR *n*

Remarks: Outputs the character whose ASCII code is *n*. Appendix D contains a chart of all ASCII codes. The *n* must be an integer from 0 to 255. If you give an input that cannot be an ASCII code, this error message appears:

CHAR DOESN'T LIKE *n* AS INPUT

Note: Any ASCII code can be entered by pressing the Alt key and holding it down while entering the ASCII code number from the numeric keypad.

Examples: TO LOWERCASE :LETTER
IF ASCII :LETTER < 64 [OP :LETTER]
IF ASCII :LETTER > 90 [OP :LETTER]
OP CHAR (ASCII :LETTER) + 32
END

This procedure outputs your input in lowercase. If you type a character other than a capital letter, this procedure outputs the character you typed.

?PRINT LOWERCASE "A
a

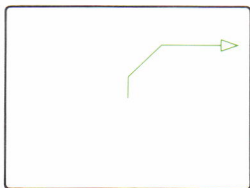
?PRINT LOWERCASE "g
g

CLEAN Command

Format: CLEAN

Remarks: Erases the graphics screen but doesn't change the turtle's position or heading. Any text displayed in the text portion of the screen is unaffected.

Examples:



?FD 15 RT 45

?FD 25 RT 45

?FD 35



?CLEAN

CLEARSCREEN (CS)

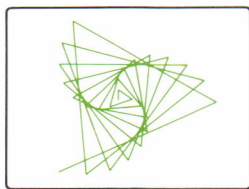
Command

Format: CLEARSCREEN

Remarks: Erases the graphics screen, puts the turtle at position [0 0] (which is the center of the screen), and sets the turtle's heading to 0 (north). If there is text on the screen, it stays as is.

Examples:

```
TO SPI :SIDE :ANGLE :INC
  IF :SIDE > 110 [STOP]
  FD :SIDE
  RT :ANGLE
  SPI :SIDE + :INC :ANGLE :INC
END
```



?SPI 8 125 4




?CLEARSCREEN

CLEARTEXT (CT)

Command

Format: CLEARTEXT

Remarks: Clears the text from the screen and puts the cursor at the upper left corner of the text part of the screen.

Note: When CLEARTEXT is used in the textscreen mode, it has the side-effect of making the function keys, F2 (mixedscreen) and F4 (fullscreen), ineffective. This is because pressing either of these keys would put the cursor outside of the text portion of the graphics screen. If you press the Enter  key until the cursor is on one of the bottom six lines of the screen, F2 and F4 will switch to the graphics mode. CLEARTEXT does *not* affect the MIXEDSCREEN and FULLSCREEN commands; these commands are always effective.

CLEARTEXT (CT) Command

Examples:

```
?REPEAT 5 [PR "HELLO]  
HELLO  
HELLO  
HELLO  
HELLO  
HELLO  
? ■
```

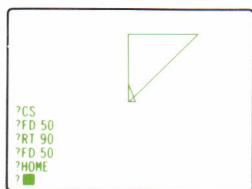
```
?REPEAT 5 [PR "HELLO]  
HELLO  
HELLO  
HELLO  
HELLO  
HELLO
```

```
? ■
```

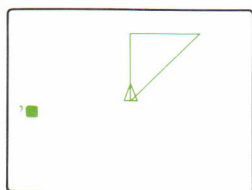
```
?CLEARTEXT
```

CLEARTEXT (CT)

Command



?MS
?CS
?FD 50
?RT 90
?FD 50
?HOME



?CLEARTEXT

CLOSE Command

Format: `CLOSE device`
 `CLOSE filespec`

Remarks: The *device* is any valid device and *filespec* is any valid filespec as explained under “Naming Files” in Chapter 4. This command closes the given file or device that is currently open. See the OPEN command to open a file or device. If you try to use CLOSE with a file or device that is not open, Logo prints the error message:

`FILE filespec IS NOT OPEN`

You cannot use CLOSE with a file that is opened by the DRIBBLE command. If you try to do so, Logo prints the error message:

`CAN'T CLOSE A DRIBBLE FILE`

When you leave Logo to enter DOS, using the .DOS command, Logo will close all files automatically.

Before you switch off your computer, be sure to close the files that are currently open. Otherwise, you will lose the data that you are currently writing to a file.

CLOSE

Command

Examples: ?CLOSE "LPT1

closes the printer.

?CLOSE "B:TELNOS

closes the file called TELNOS in drive B.

In the next example, the STORE procedure opens a file, sends data to it, and closes the file.

```
TO STORE :FILE :DATA
OPEN :FILE
SETWRITE :FILE
PRINT :DATA
CLOSE :FILE
END
```

?STORE "TELNOS [MEL: 555-4201]

The name and telephone number were written to the file called TELNOS.

CLOSEALL

Command

Format: CLOSEALL

Remarks: Closes all files and devices that are currently open.

DRIBBLE files are not closed with CLOSEALL. See the NODRIBBLE command for closing DRIBBLE files.

See the OPEN and CLOSE commands to open and close one file or device at a time. You will not receive an error message if you try CLOSEALL when no files or devices are open.

Example: ?OPEN "LPT1
?OPEN "B:TEAMLIST

You opened the printer and a file on drive B called TEAMLIST. After sending data to the file and the printer, you can close both by typing

?CLOSEALL

CO

Command

Format: CO
CO *object*

Remarks: Stands for “COntinue”.

Resumes a procedure after a PAUSE or after pressing the F5 function key, continuing from wherever the procedure paused.

If CO has an input, it becomes the output from PAUSE. This is useful when a READLIST or READCHAR has been interrupted by the F5 key, because CO's input is then read by the READLIST or READCHAR.

If you try to use CO inside a procedure, Logo prints the error message:

CO CAN'T BE USED IN A PROCEDURE

Example: The COUNTUP procedure counts forever.

```
TO COUNTUP :N
  PRINT :N
  COUNTUP :N + 1
END
```

When you run it, press F5 to make the procedure pause.

```
?COUNTUP 1
1
2
3
4
PAUSING . . . IN COUNTUP:
PRINT :N
COUNTUP?
```

The message tells you that Logo is pausing on the line `PRINT :N` in `COUNTUP`. Notice that the ? (prompt) is after the word `COUNTUP`. If you want, you can ask Logo to do something while pausing:

```
COUNTUP? PRINT 23 * 5
115
COUNTUP?
```

To get Logo to resume running the procedure, type `CO`.

```
COUNTUP? CO
5
6
7
8
.
.
.
```

If you want to stop it, just press Ctrl-Break.

CO Command

The following is an example of using CO with an input:

```
TO ECHO  
PRINT READLIST  
END
```

```
?ECHO
```

Press the F5 function key.

```
PAUSING  
JUST BEFORE LEAVING ECHO  
ECHO ?CO [HERE IS A LIST]  
HERE IS A LIST
```

COPYDEF

Command

Format: COPYDEF *newname name*

Remarks: Copies the definition of *name*, making it the definition of *newname* as well. The *name* or *newname* is the name of a procedure or a primitive. The copied definition can be saved on a disk file if *name* or *newname* is not a primitive.

COPYDEF “NEWSQUARE “SQUARE gives NEWSQUARE the same definition as SQUARE.

Note that if an error occurs when running NEWSQUARE, the error message refers to SQUARE (the old procedure name) rather than NEWSQUARE (the new procedure name).

Neither *name* nor *newname* may be names of Logo primitives unless :REDEFP is TRUE. REDEFP is a special word.

COPYDEF

Command

COPYDEF "F "FORWARD gives F the same definition as FORWARD. If :REDEFP is TRUE and *name* is a primitive, then *newname* is a primitive as well. If *newname* is a primitive, that primitive no longer exists as it used to. You may be obliged to restart Logo to continue. For example,

```
MAKE "REDEFP "TRUE
COPYDEF "BK "SQUARE
```

copies the definition of SQUARE onto BK and you will lose the primitive BK. That is why a special variable, REDEFP, is there to prevent accidental changes. So, make sure to turn the protection on by typing

```
MAKE "REDEFP "FALSE
```

after using COPYDEF with a primitive.

Comments: The procedure FEATURE redefines F to mean FD, and then tries to F 30.

```
TO FEATURE
MAKE "REDEFP "TRUE
COPYDEF "F "FD
F 30
MAKE "REDEFP "FALSE
END
```

COPYDEF Command

This procedure will not work the first time, but it will work the second time and thereafter, because of the way Logo does its parsing.

When FEATURE is first parsed, Logo thinks that F has no inputs. You redefine F, but Logo still thinks that F should have no inputs.

F complains that it is not getting any input.

If you execute FEATURE again, it will be reparsed and will work correctly.

The solution is to recover from the error, and try again.

```
TO META  
  IGNORE ERROR  
  CATCH "ERROR [FEATURE]  
  IF NOT EMPTY? ERROR [FEATURE]  
END
```

```
TO IGNORE :INPUT  
END
```

COS

Operation

Format: $\text{COS } a$

Remarks: Outputs the cosine of an angle of a degrees. The result is rounded to **PRECISION** significant digits, where **PRECISION** must be less than or equal to 100. The number a may be positive, zero, or negative.

Note that when **PRECISION** is greater than 100, Logo prints the error message:

`COS DOESN'T LIKE PRECISION n`

Example: `?MAKE "A 36045`
 `?PRINT COS :A`
 `0.7071067811`

`?PRINT COS 45`
`0.7071067811`

`?PRINT :A`
`36045`

`?PRINT COS 90`
`0`

`?SETPRECISION 30`
`?MAKE "C COS 30`
`?PRINT :C`
`0.866025403784438646763723170753`

`?SETPRECISION 150`
`?PRINT COS 67`
`COS DOESN'T LIKE PRECISION 150`

Here is a definition of the tangent function:

```
TO TAN :ANGLE
OUTPUT (SIN :ANGLE) / COS :ANGLE
END
```

```
?PRINT TAN 45
1
```

Comments: The other standard trigonometric functions (tan, cotan, sec, and cosec) may be computed by using the following expressions involving SIN and COS:

```
QUOTIENT SIN :A COS :A (for tangent of :A)
QUOTIENT COS :A SIN :A (for cotangent of :A)
QUOTIENT 1 COS :A (for secant of :A)
QUOTIENT 1 SIN :A (for cosecant of :A)
```

You may numerically check the validity of certain mathematical identities such as

$$\sin^2(x) + \cos^2(x) = 1$$

$$\cos(2x) = \cos^2(x) - \sin^2(x)$$

$$\sin(x+y) = \sin(x) \cos(y) + \cos(x) \sin(y)$$

$$\cos(x+y) = \cos(x) \cos(y) - \sin(x) \sin(y)$$

COS

Operation

This can be done by comparing, for various choices of :X and :Y and various precisions, the following:

$(\text{SIN } :X) * (\text{SIN } :X) + (\text{COS } :X) * (\text{COS } :X)$ with 1

$\text{COS } (2 * :X)$ with
 $(\text{COS } :X) * (\text{COS } :X) - (\text{SIN } :X) * (\text{SIN } :X)$

$\text{SIN}(:X + :Y)$ with
 $(\text{SIN } :X) * (\text{COS } :Y) + (\text{COS } :X) * (\text{SIN } :Y)$

$\text{COS}(:X + :Y)$ with
 $(\text{COS } :X) * (\text{COS } :Y) - (\text{SIN } :X) * (\text{SIN } :Y)$

Because these identities involve more than one operation in succession, roundoff errors may occur. Accordingly, there may be a slight difference in the last digits of the outputs when they are compared.

COUNT Operation

Format: COUNT *object*

Remarks: Outputs the number of elements in a word or list.

Example: ?PRINT COUNT [A QUICK BROWN FOX]
4

```
?PRINT COUNT [A [QUICK BROWN] FOX]
3
```

```
?PRINT COUNT "COMPUTER
8
```

```
?MAKE "CLASS [SHARNEE JENNY BELINDA JOH→
ANNE BRIAN GUY ART MICHEL PATRICK]
```

```
?PRINT COUNT :CLASS
9
```

The following procedure prints a random element of its input:

```
TO RANPICK :DATA
PRINT ITEM (1 + RANDOM COUNT :DATA) :DA→
TA
END
```

```
?RANPICK :CLASS
```

(see the list in “CLASS above)

```
BRIAN
```

```
?RANPICK "COMPUTER
M
```

CURSOR

Operation

Format: CURSOR

Remarks: Outputs a list consisting of the line and column numbers of the cursor position. The upper-left corner of the screen is [0 0] and the lower-left corner is [24 0]. The line number of the lower-right corner is 24 and the column number is the value of WIDTH-1. See the SETCURSOR command.

Example: The procedure TAB “tabs” over to the next tab stop after something has been typed. Tab stops are located in every eighth column. Note that CHAR 32 is a space.

```
TO TAB
TYPE CHAR 32
IF (REMAINDER LAST CURSOR 8) > 0 [TAB]
END
```

```
TO FLAVORCHART
TYPE "FLAVOR TAB TAB PRINT "RATING PRIN→
T [ ]
TYPE "CHOCOLATE TAB PRINT 97
TYPE "STRAWBERRY TAB PRINT 73
TYPE "BANANA TAB TAB PRINT 19
END
```

```
?FLAVORCHART
FLAVOR          RATING

CHOCOLATE       97
STRAWBERRY      73
BANANA          19
```

DEFINE Command

Format: **DEFINE** *name list*

Remarks: **DEFINE** "SQUARE [[SIDE] [REPEAT 4 [FD :S→
IDE RT 90]]]

defines the same procedure as

```
TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END
```


DEFINE makes *list* the definition of the procedure *name*. The first element of *list* is a list of the inputs to *name*, with no : (colon) before their names. If *name* has no inputs, this must be the empty list. Each subsequent element is a list consisting of one line of the procedure definition. (This list does not contain **END**, because **END** is not part of the procedure definition.)

The second input to **DEFINE** has the same form as the output from **TEXT**.

You can change the definition of a Logo primitive by using **DEFINE**. **DEFINE** won't work for primitives unless the special word, **REDEFP**, is **TRUE**. (It is initially **FALSE**.)

If you are defining a new function inside a procedure for immediate execution, then see "Comments" under the **COPYDEF** command.

DEFINE Command

Example: LEARN is a program that lets you type successive lines defining a procedure that has no inputs. Each time you press the Enter  key, Logo runs the instruction as well as making it part of the procedure definition. By typing ERASE, you can erase the previous line.

```
TO LEARN
MAKE "PRO [[]]
READLINES
PR [DO YOU WANT TO KEEP THIS AS A PROCEDURE?]
IF FIRST READWORD = "N [STOP]
PR []
TYPE [PROCEDURE NAME ?]
DEFINE READWORD :PRO
END

TO READLINES
MAKE "NEXTLINE READLIST
IF :NEXTLINE = [END] [PR [ ] STOP]
TEST :NEXTLINE = [ERASE]
IFTRUE [CANCEL]
IFFALSE [RUN :NEXTLINE MAKE "PRO LPUT :>
NEXTLINE :PRO]
READLINES
END

TO CANCEL
PR SE [I WILL ERASE LINE] LAST :PRO
MAKE "PRO BL :PRO
END
```

DEFINE Command

```
?LEARN  
FD 20  
RT 36  
ERASE  
I WILL ERASE LINE RT 36  
RT 72  
END
```

```
DO YOU WANT TO KEEP THIS AS A PROCEDURE →  
?  
YES  
PROCEDURE NAME ?LEG
```

```
?PO "LEG  
TO LEG  
FD 20  
RT 72  
END
```



```
?LEG
```

DEFINE

Command

You must make REDEFP TRUE to change the definition of FD.

```
MAKE "REDEFP "TRUE
DEFINE "FD [[N] [BK :N]]
```

Type the following:

```
FD 100
```

Later you could restore FD's definition, by typing

```
COPYDEF "FD "FORWARD
```

One possible reason for using DEFINE is to get information about the running of your program to help in debugging. For example, if you find that your variables have the wrong values, you may find it helpful to have a version of MAKE that records what it's doing. Here's how:

```
?COPYDEF "TRUE.MAKE "MAKE
?DEFINE "MAKE [[NAME THING] [PR (SE "MA→
KING :NAME "\=:THING)] [TRUE.MAKE :NAM→
E :THING]]
```

```
?MAKE "JOB 259
MAKING JOB = 259
```

After you have redefined primitives, you may want to make :REDEFP FALSE to protect the primitives from accidental changes.

DEFINEDP Operation

Format: DEFINEDP *word*

Remarks: Outputs TRUE if *word* is the name of a procedure;
 otherwise FALSE.

Example: In the program called LEARN in the previous
 section (under DEFINE), a procedure defined
 previously can be erased if you give the same name
 again. You can use DEFINEDP in the procedure
 LEARN to avoid it.

```
TO LEARN
MAKE "PRO [ ]
READLINES
PR [DO YOU WANT TO KEEP THIS AS A PROCEDURE?]
IF FIRST READWORD = "N [STOP]
PR [ ]
TYPE [PROCEDURE NAME ?]
MAKE "NAME READWORD
TEST DEFINEDP :NAME
IFT [PR [ ] TYPE [PROCEDURE ALREADY EXISTS. CHOOSE ANOTHER NAME :] DEFINE READWORD :PRO]
IFF [DEFINE :NAME :PRO]
END
```

```
TO READLINES
MAKE "NEXTLINE READLIST
IF :NEXTLINE = [END] [PR [ ] STOP]
TEST :NEXTLINE = [ERASE]
IFTRUE [CANCEL]
IFFALSE [RUN :NEXTLINE MAKE "PRO LPUT :NEXTLINE :PRO]
READLINES
END
```


DEFINEDP

Operation

```
TO CANCEL  
PR SE [I WILL ERASE LINE] LAST :PRO  
MAKE "PRO BL :PRO  
END
```

```
?LEARN  
FD 60  
RT 120  
FD 30  
RT 150  
FD 25  
HIDETURTLE  
END
```

```
DO YOU WANT TO KEEP THIS AS A PROCEDURE→  
?  
YES
```

```
PROCEDURE NAME ?LEARN
```

```
PROCEDURE ALREADY EXISTS. CHOOSE ANOTHE→  
R NAME: FLAG
```

```
?PO "FLAG  
TO FLAG  
FD 30  
RT 120  
FD 30  
RT 150  
FD 25  
HIDETURTLE  
END
```

DIFFERENCE

Operation

Format: DIFFERENCE *a b*

Remarks: Outputs the result of subtracting *b* from *a*. DIFFERENCE is equivalent to $-$ (minus), an infix operation. If one of the inputs is missing, this error message appears:

```
NOT ENOUGH INPUTS TO DIFFERENCE
```

Example:

```
?PRINT DIFFERENCE 7 1
6
?PRINT DIFFERENCE 13.23 5.1
8.13
?PRINT DIFFERENCE (5 + 6) (3 * 7)
-10
```

The procedure GUESSNUM asks you to guess a number. Then it prints the absolute difference between the number you guessed and the random number picked by Logo.

```
TO GUESSNUM
  MAKE "NUM RANDOM 10
  PRINT [GUESS A NUMBER]
  MAKE "GUESS READWORD
  TEST EQUALP :GUESS :NUM
  IFTRUE [PRINT [YOU GOT IT!]]
  IFFALSE [(PRINT [YOU ARE] ABS DIFFERENCE
  E :GUESS :NUM [FROM THE NUMBER])]
  END

TO ABS :NUM
  OP IF :NUM < 0 [-:NUM] [:NUM]
  END
```

DIR

Command

Format: DIR
DIR *drive*:
DIR *filespec*

Remarks: Stands for “DIRectory”.

DIR lists all files on the disk in the default drive. See the DISK command.

Each disk file is listed according to its file name, its extension, the file size in bytes, and the most recent date and time the file was altered or created.

The *drive* input allows you to specify the drive from which you want the directory:

?DIR "B:

lists the directory from drive B.

The *filespec* input allows you to specify which file name and extension you want to list.

?DIR "GAME.LF

confirms the existence on the disk of your Logo program file called GAME.LF by returning the directory information about it. This lets you check on a file without reading the entire directory. If the file does not exist, Logo prints the error message:

FILE *filespec* DOES NOT EXIST

Whenever you omit the *drive*, the default drive is assumed.

DIR Command

Two special characters, ? and *, give you greater flexibility with the DIR command. They can be used within a file name and its extension. The * must be preceded by a backslash. (Otherwise Logo would try to multiply.)

The ? in a file name or extension indicates that any character can take that position.

`DIR "TE?T.DAT`

lists information on all files having four characters beginning with TE, any next character, T, and an extension of DAT. For example, TEST.DAT and TEXT.DAT would be listed.

The * in a file name or extension indicates that any character can occupy that position and all remaining positions in a file name or extension.

`?DIR "*.LF`

lists information only on those files with .LF extensions on the disk in the default drive (see the DISK operation).

`DIR "TE*.DAT`

lists all directory entries on the default drive that have file names beginning with TE and an extension of DAT. In this case, the file names may be from two to eight characters in length. For example, TEST.DAT, TECO.DAT, and TESTLOGO.DAT would be listed.

DISK

Operation

Format: DISK

Remarks: Outputs the default drive (A, B, C, etc). A two-drive system has the drives A and B. When Logo begins, the default drive is the current DOS default. See the SETDISK command to change the default drive.

Example: ?PRINT DISK
 A:

 ?SETDISK "B:

 ?PRINT DISK
 B:

DOT Command

Format: DOT *position*

Remarks: Puts a dot of the current pen color at the specified *position*, without moving the turtle. The *position* is a list of the x-coordinate and y-coordinate. DOT does not draw a line, even if the pen is down.

DOT [160 0] puts a dot halfway down the right edge of the screen.



CS

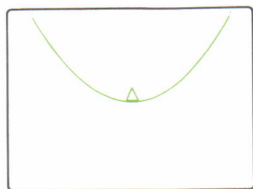


DOT [160 0]

Example: The procedure CURVE will draw a parabola if its input is -100.

:X * :X / 100 gives the Y value for each X.

```
TO CURVE :X
IF :X = 100 [STOP]
DOT LIST :X :X * :X / 100
CURVE :X + 1
END
```



?CURVE -100

DRIBBLE

Command

Format: DRIBBLE *device*
 DRIBBLE *filespec*

Remarks: The *device* is any valid device and *filespec* is any valid file as explained under “Naming Files” in Chapter 4. DRIBBLE starts the process of sending a copy of most of the characters displayed on the textscreen to *device* or *filespec*. The characters which cannot be dribbled are indicated under “Dribble File” in Chapter 4. DRIBBLE records interactions between the IBM Personal Computer and the person at the keyboard. DRIBBLE automatically opens *filespec*.

You cannot DRIBBLE to the screen, as the following shows:

```
DRIBBLE "CON
CAN'T DRIBBLE ON SCREEN
```

The command NODRIBBLE stops the process of dribbling. If you try to use the command CLOSE, Logo prints the error message:

```
CAN'T CLOSE A DRIBBLE FILE
```


DRIBBLE Command

You cannot SETREAD or SETWRITE to a dribble file while still dribbling. Logo prints the error message:

FILE filespec ALREADY SELECTED FOR DRIBBLE

However, once a dribble file on disk has been closed with NODRIBBLE, it may be processed like any other Logo file. You can then open the file, read from it, or write to it.

Note that only one dribble file can be open at one time. See Chapter 4, “File Handling” for a more detailed discussion of dribble files.

Example: ?DRIBBLE "JUNE1

creates a file called JUNE1 and starts the dribbling process. Every line appearing after this on the screen is sent to this file:

```
?CS  
?FD 100  
?RT 80  
?FD 50  
?NODRIBBLE
```

DRIBBLE

Command

To print the contents of a dribble file onto a printer, the following DUMP procedure can be used:

```
TO DUMP :FILENAME  
DRIBBLE "LPT1  
POFILE :FILENAME  
NODRIBBLE  
END
```

```
?DUMP "JUNE1  
CS  
FD 100  
RT 80  
FD 50  
NODRIBBLE
```

To print the dribble file to a serial printer:

```
? .SETCOM 1 300 2 7 1  
?OPEN "COM1  
?DRIBBLE "COM1
```

Note that the inputs to .SETCOM depend on your particular serial printer. See the .SETCOM command and your printer manual.

EDIT (ED) Command

Format: EDIT
 EDIT *name*
 EDIT *list*

Remarks: Enters the Logo editor. If an input is given, the editor starts up with the definitions of the given procedures in the edit buffer. If any procedure *name* has not been previously defined, the edit buffer contains the title line (TO *name*) and END. If no input is given, the edit buffer has the same contents that it did the last time you used the editor.

Pressing the Esc key is the standard way to complete the definition and exit from the editor. Logo reads every line from the edit buffer as though you had typed it outside the editor.

If the end of the buffer is reached while there is a procedure definition in the editor. Logo completes the procedure definition and inserts END.

The procedures that have been defined stay in the workspace but will disappear when the machine is switched off. See the SAVE command to store procedures permanently.

EDIT (ED) Command

Pressing Ctrl-Break interrupts editing. Use it if you don't like the changes you are making or you decide not to make changes. Any changes you made in the edit buffer will be ignored. If you were editing a procedure, the definition will be the same as it was before you started editing.

If you press Ctrl-Break when you don't mean to, you can get back to your edit buffer by typing EDIT (no input).

When you are in the editor, you may use all of the editing actions.

See Chapter 3, "Logo Editor," for more information.

EDITFILE

Command

Format: EDITFILE *filespec*
EDITFILE *infilespec outfilespec*

Remarks: EDITFILE may be used with one input, *filespec*. In this case, the edited contents will be saved under the same file name, and the old contents will be lost.

EDITFILE also can be used with two inputs.

The *infilespec* is a valid, existing file name, as explained under “Naming Files” in Chapter 4. EDITFILE loads the file named by *infilespec* into the edit buffer. You can use EDITFILE on any file without opening it previously. The file cannot exceed 4096 bytes. If you try EDITFILE with a file longer than the maximum, Logo prints the error message:

EDITOR BUFFER IS FULL

The *outfilespec* is the name given to the file once it is saved onto the disk. (The *infilespec* still exists.) If *outfilespec* already exists, you will get the error message

FILE *filespec* ALREADY EXISTS

before you enter the editor.

EDITFILE

Command

When you have finished editing the file by pressing the Esc key, the file will be saved onto the disk with the name given by *outfilespec*. If you get the error message

DISK FULL

when you exit from the editor, you can change diskettes and recover the changes you have made by typing

EDITFILE "EDITOR *outfilespec*

EDITOR is a special word you can use to edit the current contents of the edit buffer.

EDITFILE may be used to edit an ASCII file, but not a binary file.

For further information on how to use the Logo editor, see Chapter 3, "Logo Editor."

7-101

EDNS

Command

Examples: ?PONS
ANIMAL IS GIBBON
SPEED IS 55
AIRCRAFT IS [JET HELICOPTER]
?EDNS

The editor is entered with the following information:

MAKE "ANIMAL "GIBBON
MAKE "SPEED 55
MAKE "AIRCRAFT [JET HELICOPTER]

If you edit to make the following changes

MAKE "ANIMAL "GRYFFIN
MAKE "SPEED 55
MAKE "AIRCRAFT [JET HELICOPTER BLIMP]

and exit the editor by pressing the Esc key, the result is:

?PONS
ANIMAL IS GRYFFIN
SPEED IS 55
AIRCRAFT IS [JET HELICOPTER BLIMP]

EFORM Operation

Format: EFORM n a

Remarks: Outputs the number n in scientific notation, using a digits. The output is rounded to a digits, where a must be a positive, real number from 1 through 1000. If a is not an integer, it is rounded to the nearest integer.

Examples: ?PR EFORM 12345.0006789 6
1.23450E+0004

?PR EFORM -643.27265004 20/6
-6.43E+0002

?MAKE "A -1.234567845
?PR EFORM :A 8
-1.2345678E+0000

?SETPRECISION 9
?PR EFORM :A 8
-1.2345679E+0000

EMPTY

Operation

Format: EMPTY *object*

Remarks: Outputs TRUE if *object* is the empty word or the empty list; otherwise, FALSE.

Examples: ?PRINT EMPTY 3
 FALSE

 ?PRINT EMPTY BUTFIRST "UNICORN
 FALSE

 ?PRINT EMPTY BUTLAST "U
 TRUE

 ?PRINT EMPTY BUTFIRST [UNICORN]
 TRUE

The procedure, TALK, matches animal sounds to animals.

```
TO TALK :ANIMALS :SOUNDS
IF OR EMPTY :SOUNDS EMPTY :ANIMALS [P→
R [THAT'S ALL THERE IS!] STOP]
PR SE FIRST :ANIMALS FIRST :SOUNDS
TALK BF :ANIMALS BF :SOUNDS
END
```

```
?TALK [DOGS BIRDS PIGS] [BARK CHIRP OINK]
K]
DOGS BARK
BIRDS CHIRP
PIGS OINK
THAT'S ALL THERE IS!
```

EMPTYP Operation

REVPRINT prints its input in reverse.

```
TO REVPRINT :THING
IF EMPTYP :THING [PR [] STOP]
TYPE LAST :THING
IF LISTP :THING [TYPE CHAR 32]
REVPRINT BL :THING
END

?REVPRINT "ELEPHANT
TNAHPELE
?REVPRINT "PUMPERNICKEL
LEKCINREPMUP
?REVPRINT [ALISON LOVES MATTHEW]
MATTHEW LOVES ALISON
?REVPRINT "12345
54321
```

EQUALP

Operation

Format: `EQUALP object1 object2`

Remarks: Outputs TRUE if *object1* and *object2* are equal numbers, identical words, or identical lists; otherwise, outputs FALSE. Equivalent to = (equal), an infix operation.

Examples: `?PRINT EQUALP "RED FIRST [RED YELLOW]`
 TRUE

`?PRINT EQUALP 100 50 * 2`
TRUE

`?PRINT EQUALP [THE A AN] [THE A]`
FALSE

`?PRINT EQUALP " []`
FALSE

(The empty word and the empty list are not identical.)

EQUALP Operation

The following operation tells whether its first input is an element of its second input. This is similar to the primitive MEMBERP.

```
TO INP :ONE :ALL
  IF EMPTY :ALL [OUTPUT "FALSE]
  IF EQUALP :ONE FIRST :ALL [OUTPUT "TRUE]
  OUTPUT INP :ONE BUTFIRST :ALL
END
```

```
?PRINT INP "S "PERSONAL
TRUE
```

```
?PRINT INP "PERSON "PERSONAL
FALSE
```

```
?PRINT INP "ON [IN ON AT]
TRUE
```

```
?PRINT INP "2 "12345
TRUE
```

ERALL

Command

Format: ERALL
 ERALL *package*
 ERALL *packagelist*

Remarks: When ERALL has no input, it erases all procedures and variables from the workspace except the ones that are buried. See the BURY command.

If an input *package* or *packagelist* is present, it erases only the procedures and variables in the *package(s)*, leaving the rest untouched. In this case, even buried procedures are erased as long as they are included in the *package(s)* to be erased.

Because ERALL takes an optional input, you cannot put another command immediately after ERALL on the same line. Be sure to put the next command on a new line.

Note that ERALL does not reclaim the space used by properties created by PPROP.

ERALL Command

Examples: Suppose the following procedures are in your workspace. The command POTS prints out the titles. After ERALL, POTS no longer prints them out.

```
?POTS  
TO SNAKE  
TO SLITHER
```

```
?ERALL  
?POTS  
?
```

POTS shows no procedures now, but there are some procedures buried.

```
?POTS "USEFUL  
TO POLY :SIDE :ANGLE  
TO DIVISORP :A :B
```

In order to erase these procedures, type

```
?ERALL "USEFUL
```

ERASE (ER)

Command

Format: ERASE *name*
 ERASE *namelist*

Remarks: Erases the named procedures from the workspace.
 Procedures that are in files on a disk are not affected
 by ERASE.

Even a procedure which is buried can be erased.

Examples: ERASE "TRIANGLE

 erases the TRIANGLE procedure.

 ERASE [TRIANGLE SQUARE]

 erases the TRIANGLE and SQUARE procedures.

ERASEFILE

Command

Format: ERASEFILE *filespec*

Remarks: Erases the file named *filespec* from the disk. The *filespec* is the name of a disk file. See “Naming Files” in Chapter 4 for information. If the file you want to erase has an extension, you must remember to include it. If there is no file by the name you have specified, Logo prints the error message:

FILE *filespec* DOES NOT EXIST

To see which files exist on your disk, use the DIR command.

Note that Logo program files automatically have the extension .LF when you save them with SAVE. Logo will look for the file with this extension unless another extension is specified by using a period. If you want to erase a file that doesn't have any extension, you *must* put a period at the end of your file name to specify the null extension.

ERASEFILE accepts two special characters, ? and *. This gives you greater flexibility. The * must be preceded by a \ (backslash).

The ? in a file name or extension indicates that any character can occupy that position. The * in a file name or extension indicates that any character can occupy that position and all remaining positions in a file name or extension. See the DIR command for more information on these special characters.

ERASEFILE

Command

Examples: `?ERASEFILE "BEAR`

erases the file BEAR with the extension .LF (a Logo program file).

`?ERASEFILE "BEAR.`

erases a file named BEAR (with no extension).

`?ERASEFILE "DATA.TEL`

erases the file DATA with the extension .TEL.

`?ERASEFILE "B:SYLVIE.`

erases the file SYLVIE from the diskette in drive B.

`?ERASEFILE "DATA.*`

erases all files named DATA ending with any extension from the disk in the default drive.

`?ERASEFILE "DAT?.???`

erases all the files with four-character names starting with DAT and having any extension.

Format: ERN *name*
 ERN *namelist*

Remarks: Stands for “ERase Name”.

Erases the named variables from the workspace. These variables were created with MAKE or NAME. To see which variables are in your workspace, use PONS (Print Out NameS).

Even buried variables can be erased.

Examples: ?PONS
 LENGTH IS 50
 X IS 3.1416
 NUM IS 6

?ERN "LENGTH

erases the LENGTH variable.

?ERN [X NUM]

erases the X and NUM variables.

ERNS

Command

Format: ERNS
ERNS *package*
ERNS *packagelist*

Remarks: Stands for “ERase NameS”.

Erases all variables from the workspace. When ERNS is given no input, it leaves buried names untouched. See the BURY command.

If ERNS is used with an input, only the names in the packages are erased. In this case, buried names in the packages are also erased.

Because *package* and *packagelist* are optional inputs, do not put another command on the same line as ERNS when ERNS does not have an input.

To put a variable into a package use the PPROP or PKGALL command.

Examples: ?ERNS
?PONS
?

There are no variables printed when you give the command PONS (Print Out NameS), but there are some names in a buried package.

```
?PONS "USEFUL  
LENGTH IS 50  
X IS 3.1416  
NUM IS 6
```

```
?ERNS "USEFUL
```

erases the names in the package USEFUL.

ERPS Command

Format: ERPS
ERPS *package*
ERPS *packagelist*

Remarks: Stands for “ERase ProcedureS”.

Erases all procedures from the workspace. When there is no input, ERPS does not erase buried procedures. See the BURY command.

When used with an input, ERPS erases all procedures included in that package. Buried ones are also erased in this case.

Because *package* and *packagelist* are optional inputs, do not put another command on the same line as ERPS when ERPS does not have an input.

Examples: ?ERPS
?POTS
?

There are no procedure names printed after you give the command ERPS. If you want to erase procedures buried in a package, you must give ERPS an input.

?ERPS "USEFUL

erases the procedures in the package USEFUL.

ERROR

Operation

Format: ERROR

Remarks: Outputs a six-element list containing information about the most recent error which has not had a message issued by ERROR. It outputs an empty list if there was no such error or if the error is too deep to preserve the real cause. The output list contains:

1. A unique number identifying the error (the numbers are listed in Appendix A).
2. A message explaining the error.
3. The name of the procedure within which the error occurred (the empty list, if top level).
4. The line where the error occurred.
5. If any, the name of the primitive causing the error.
6. If any, the name of the object causing the error.

An error that occurs while running a procedure normally stops all procedures and prints the error message.

An error can cause a pause without exiting from the procedure altogether. This can be very helpful for debugging purposes. To do this, just change the special variable called ERRACT by typing

`MAKE "ERRACT "TRUE`

ERROR Operation

You will then get control back at the point in the procedure that causes the error. For example,

```
TO TRY  
PR :MESSAGE  
END
```

If you run this procedure when `ERRACT` is `TRUE`, the procedure will pause. To continue, give `CO` an input for the message. Then `TRY` will execute.

```
MESSAGE HAS NO VALUE :  
JUST BEFORE LEAVING TRY  
TRY ? CO [TRY WORKS]  
TRY WORKS  
?
```

To turn off the debugging feature, type

```
MAKE "ERRACT "FALSE
```

Logo runs `THROW "ERROR` whenever an error occurs unless `:ERRACT` is `TRUE`. `ERROR` is a special word used when an error occurs. If `:ERRACT` is `FALSE` and `CATCH "ERROR` has been run at the time an error occurs, control returns there. When an error is caught in this way, no error message is printed. You can, however, design your own.

ERROR Operation

Example: TO SAFESQUARE :SIDE
CATCH "ERROR [REPEAT 4 [FD :SIDE RT 90]]
PRINT ERROR
END

```
?SAFESQUARE "SIXINCHES
41 [FORWARD DOESN'T LIKE SIXINCHES AS I→
NPUT] SAFESQUARE [CATCH "ERROR [REPEAT →
4 [FD :SIDE RT90]]] FORWARD SIXINCHES
```

SAFESQUARE runs CATCH "ERROR and prints ERROR if an error occurs. You can modify this procedure to print your own error message.

```
TO SAFESQUARE :SIDE
CATCH "ERROR [REPEAT 4 [FD :SIDE RT 90]]
PRINT [YOU MADE A BUG!]
END
```

```
?SAFESQUARE "SIX
YOU MADE A BUG!
```

EXP Operation

Format: `EXP a`

Remarks: `EXP a` outputs the number e raised to the power a . Here e is the mathematical constant used as the base of natural logarithms. The result is rounded to `PRECISION` significant digits, where `PRECISION` must be less than or equal to 100.

To obtain the constant e itself (at the current precision), just use `EXP 1`.

When `PRECISION` is greater than 100, Logo prints the error message:

`EXP DOESN'T LIKE PRECISION n`

Examples: `?SETPRECISION 20`
 `?PRINT EXP 1`
 2.7182818284590452354
 `?SETPRECISION 40`
 `?PRINT EXP 1`
 2.7182818284590452353602874713526624977→
 57
 `?SETPRECISION 150`
 `?PRINT EXP 1`
 `EXP DOESN'T LIKE PRECISION 150`

`?SETPRECISION 20`
 `?PRINT EXP 5`
 148.41315910257660342
 `?PRINT EXP -5`
 6.7379469990854670966E-0003
 `?MAKE "B 4.2163774`
 `?PRINT EXP :B`
 67.787472060223108808

EXP

Operation

```
?PRINT EXP -:B  
1.4751988377905424748E-0002  
?PRINT PRODUCT EXP :B EXP -:B  
0.9999999999999999997  
?PRINT EXP 0  
1  
?PRINT EXP -100  
3.7200759760208359630E-0044
```

This example prints :STEPS values describing the exponential decay of :START at :RATE.

```
TO DECAY :START :RATE :STEPS  
IF :STEPS < 1 [STOP]  
MAKE "NEXT :START / EXP :RATE  
PRINT :NEXT  
DECAY :NEXT :RATE :STEPS-1  
END
```

```
?DECAY 100 1 4  
36.787944117144232159  
13.533528323661269189  
4.9787068367863942977  
1.8315638888734180293
```

The WORD operation can be used to enter numbers that are longer than a Logo line.

You may wish to numerically check the validity of certain mathematical identities such as

$$e^{x+y} = e^x * e^y$$

$$(e^x)^y = e^x * y$$

This can be done by comparing for various choices of :X and :Y and various precisions:

EXP (:X + :Y) with EXP (:X) * EXP (:Y)

POWER (EXP :X) :Y with EXP (:X * :Y)

Because these identities involve more than one operation in succession, roundoff errors may occur. Accordingly, there may be a slight difference in the last digits of the outputs when they are compared.

FENCE

Command

Format: FENCE

Remarks: Fences the turtle within the edges of the screen. If an attempt is made to move the turtle beyond the edges of the screen, the turtle does not move and the error message

TURTLE OUT OF BOUNDS

is displayed. If the turtle is already off the screen when you type FENCE, an error message is displayed. In this case, the FENCE mode is not in effect. You will have to bring the turtle within the boundaries of the screen and type the FENCE command again.

The screen boundaries are:

horizontal — 159 through 160
vertical — 124 through 125.

See also the WINDOW and WRAP commands.

FENCE
CS
RT 5
FD 500

produces the error message

TURTLE OUT OF BOUNDS

FILELEN Operation

Format: FILELEN *filespec*

Remarks: The *filespec* is the file name that was used with the OPEN command. FILELEN outputs the length (in bytes) of the contents of the file called *filespec*. If you have not opened the file by using the command OPEN *filespec* before using FILELEN, Logo prints the error message:

FILE *filespec* IS NOT OPEN

Note that the length of the file specified by FILELEN is one byte less than the length shown with the command DIR. This is because FILELEN's length does not include the end-of-file character.

Examples: ?OPEN "ADDRESS

```
?PRINT FILELEN "ADDRESS
128
```

The file called ADDRESS already has data 128 bytes long.

```
TO FILLIN :FILE :LEN
  OPEN :FILE
  SETWRITE :FILE
  MAKE "SPACE :LEN - FILELEN :FILE
  IF :SPACE > 0 [REPEAT :SPACE [TYPE 0]]
  CLOSE :FILE
END
```

The procedure FILLIN opens the file :FILE and fills it in with zeros so the file will be :LEN bytes long.

FILEP

Operation

Format: FILEP *filespec*

Remarks: The *filespec* is a valid file name as explained under “Naming Files,” in Chapter 4.

Outputs TRUE if a file called *filespec* exists on the disk; otherwise FALSE.

Examples: ?PRINT FILEP "HANOI.LF
FALSE

The file called HANOI.LF does not exist.

```
TO WRITEDATA :FILE :DATA
IF FILEP :FILE [PRINT [THIS FILE ALREADY EXISTS!] STOP]
OPEN :FILE
SETWRITE :FILE
PRINT :DATA
CLOSE :FILE
END
```

```
?WRITEDATA "TELNOS [JOANNE: 555-1414]
THIS FILE ALREADY EXISTS!
```

You have decided that you do not want to write to a file that already exists, so you include the first line in the WRITEDATA procedure. If you try WRITEDATA with an already existing file, you get a message on the screen. If you remove the first line of the WRITEDATA procedure, you can use it with any file or device that you can write to.

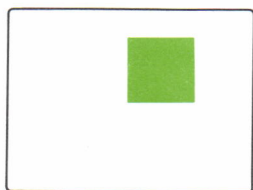
FILL Command

Format: FILL

Remarks: Fills a closed shape, which has the turtle in it, with the current pen color. If the turtle is not within the interior of the closed shape, set the turtle's pen state to pen up, move the turtle inside the shape, put the pen down, and then fill with color.

This command can be used only if you have an IBM Color/Graphics Monitor Adapter.

Example: `TO FILLSQ
REPEAT 4 [FD 50 RT 90]
SETPC 2
PU FD 25 RT 90 FD 25 PD
FILL
END`



`?FILLSQ`

The entire square is colored in magenta.

FILL

Command

If the turtle is on a horizontal or vertical line, **FILL** redraws the line and all connected horizontal and vertical lines in the current pen color. If the turtle is on a slanted line, **FILL** has no effect.

```
?REPEAT 4 [FD 50 RT 90]  
?SETPC 2 FILL
```

The color of the outline of the square changes to magenta.

Comments: Some shapes with interior angles greater than 180° may not **FILL** completely. Try moving the turtle to another position within the figure or dividing the figure into smaller sections, and filling each separately.

FIRST Operation

Format: **FIRST** *object*

Remarks: Outputs the first element of *object*. **FIRST** of an empty word or of an empty list produces the error message:

FIRST DOESN'T LIKE *object* AS INPUT

Note that **FIRST** of a word is a single character;
FIRST of a list can be a word or a list.

Examples: **?PRINT FIRST [HOUSE MOUSE LOUSE]**
HOUSE

?PRINT FIRST "HOUSE
H

?PRINT FIRST [HOUSE]
HOUSE

?SHOW FIRST [[THE A AN] [UNICORN RHINO]→
[SWIMS FLIES GROWLS RUNS]]
[THE A AN]

?PRINT FIRST "
FIRST DOESN'T LIKE AS INPUT

?PRINT FIRST []
FIRST DOESN' T LIKE [] AS INPUT

?PRINT FIRST .010
0

?PRINT FIRST ".010

FIRST

Operation

The procedure PRINTDOWN prints each element of its input on a separate line.

```
TO PRINTDOWN :INPUT
IF EMPTY? :INPUT [STOP]
PR FIRST :INPUT
PRINTDOWN BF :INPUT
END

?PRINTDOWN "MOUSE
M
O
U
S
E
?PRINTDOWN [A STRAWBERRY SUNDAE]
A
STRAWBERRY
SUNDAE
```

FORM Operation

Format: `FORM n a`
 `FORM n a b`

Remarks: `FORM` outputs the number n with a digits before the decimal, and b digits after it. The input for a or b must be a number from 0 through 1000. If the input to b is omitted, the output of n has no digits after the decimal point, and the decimal point is not printed.

If the integer part of n has fewer digits than a , spaces are added preceding the number, and n uses $a + b + 1$ places (1 for the decimal point). If n is a negative number, the $-$ (minus) sign takes one place.

If a is less than the number of digits before the decimal point in n , or if a is 0, the number of digits before the decimal point is unchanged. If b is 0, `FORM` has the same effect as when b is omitted. If b is greater than the number of digits after the decimal point in n , trailing zeros are added.

`FORM` is useful when you are trying to print columns of numbers in an unvarying format.

`FORM` rounds numbers a and b to the nearest integer. See the `EFORM` command to change the form of numbers in scientific notation. See the `SETPRECISION` command to change the precision of numbers. For further information on precision, see “Precision and Logo Numbers” in Chapter 5.

FORM

Operation

Examples: ?PR FORM 12345.0006789 6 4
12345.0007
?MAKE "A -1.234567845
?PR FORM :A 2 7
-1.2345678
?PR FORM 1234567890123456789 0 0
123456789000000000

Don't forget we are in PRECISION 10.

?PR FORM 24.8321 3 20/3
24.8321000

The *b* is rounded to 7.

?SETPRECISION 9
?PR FORM :A 2 7
-1.2345679
?PR :A
-1.23456785

The last series of examples illustrates how double roundings can take place when you change precision. The value of :A in your workspace is equal to -1.234567845 (entered when in precision 10). Then you switch to precision 9. According to the rules of precision, the value of :A is not changed. When you do FORM :A 2 7, Logo first rounds :A to nine significant digits. FORM uses this value to complete the operation. Therefore, Logo does

FORM -1.23456785 2 7

FORM Operation

FORM rounds the number -1.23456785 to seven decimal places, thereby making -1.2345679 the output of FORM.

Here is a procedure to format numbers with dollar signs.

```
TO PRDOLLARS :N  
TYPE "$"  
PRINT FORM :N 7 2  
END
```

```
?PRDOLLARS 15 * .15  
$ 2.25
```

```
?PRDOLLARS 100.00 * 9  
$ 900.00
```

FORWARD (FD)

Command

Format: FORWARD *distance*

Remarks: Moves the turtle forward *distance* steps in the direction in which it is heading. The *distance* is any real number from -9999 through 9999. If you try to move the turtle beyond the boundary of the screen, one of the following will happen:

- In FENCE mode the turtle does *not* move, and Logo prints the error message:

TURTLE OUT OF BOUNDS

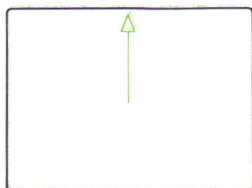
- In WINDOW mode the turtle disappears off the edge of the screen.
- In WRAP mode the turtle reappears from the opposite side of the screen.

FORWARD (FD) Command

Examples:

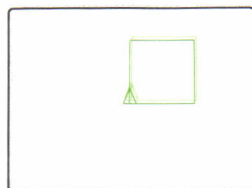


?CS



?FORWARD 70

```
TO SQUARE :SIDE  
REPEAT 4 [FORWARD :SIDE RIGHT 90]  
END
```



?CS

?SQUARE 50

FPUT

Operation

Format: FPUT *object list*

Remarks: Stands for “First PUT”.

Outputs a new list formed by putting *object* at the beginning of *list*. See the LIST operation for a chart comparing FPUT with other primitives that combine words and lists.

Examples: ?SHOW FPUT "HAMSTER [DOG CAT]
[HAMSTER DOG CAT]

?SHOW FPUT [THE A AN] [CUP GLASS]
[[THE A AN] CUP GLASS]

?SHOW FPUT "A []
[A]

The following procedure adds a rhyming word to a list of words.

```
TO ADD.RHYME
PR [ADD A RHYMING WORD TO THIS LIST:]
PR :RHYMES
MAKE "NEWRHYME READWORD
MAKE "RHYMES FPUT :NEWRHYME :RHYMES
PR SE [THE NEW RHYMING WORDS ARE:]
PR :RHYMES
END
```

```
?MAKE "RHYMES [MEND LEND]
?ADD.RHYME
ADD A RHYMING WORD TO THIS LIST:
MEND LEND
DEFEND
THE NEW RHYMING WORDS ARE:
DEFEND MEND LEND
```

FULLSCREEN (FS) Command

Format: FULLSCREEN

Remarks: Devotes the entire screen to graphics. Only the turtle graphics show. Any text you type will be invisible to you, although Logo will still carry out your instructions. The text will reappear when you return to MIXEDSCREEN or TEXTSCREEN mode.

If Logo needs to type an error message while you are in FULLSCREEN, it automatically returns to MIXEDSCREEN.

Pressing the F4 function key has the same effect as typing FULLSCREEN. In addition, the F4 key can be pressed while a procedure is running, whereas you must wait to get the ? (prompt) in order to type FULLSCREEN.

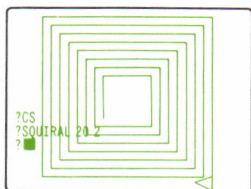
If Logo is in TEXTSCREEN mode and a graphics command is given, Logo will automatically return to either the FULLSCREEN or MIXEDSCREEN mode (whichever was used last).

However if .SCREEN is 2, neither FULLSCREEN nor the F4 function key will have any effect.

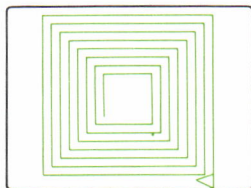
FULLSCREEN (FS)

Command

Examples: TO SQUIRAL :STEP :INC
IF :STEP > 80 [STOP]
FD :STEP
RT 90
RT 90
SQUIRAL :STEP + :INC :INC
END



?MIXEDSCREEN
?CS
?SQUIRAL 20 2



?FULLSCREEN

GO Command

Format: *GO word*

Remarks: Transfers control to the instruction following LABEL *word* in the same procedure. GO may be used to create loops within a procedure. One way to leave a loop is by executing another GO command. Another way is to execute STOP or OUTPUT, which will cause Logo to return to the calling procedure. If you put the command GO in a procedure without LABEL, Logo prints the error message:

CAN'T FIND LABEL word

Examples: TO COUNTDOWN :N
 LABEL "LOOP
 IF :N < 0 [STOP]
 PRINT :N
 MAKE "N :N - 1
 GO "LOOP
 END

?COUNTDOWN 4
4
3
2
1
0

GO

Command

The COUNTDOWN procedure has the same effect as COUNTD, a recursive procedure:

```
TO COUNTD :N
IF :N < 0 [STOP]
PRINT :N
COUNTD :N - 1
END
```

```
?COUNTD 4
4
3
2
1
0
```

GPROP Operation

Format: GPROP *name prop*

Remarks: Stands for “Get PROPerTy”.

Outputs the value of the *prop* property of *name*. To set the property, use the command PPROP. If the property does not exist, GPROP outputs the empty list. See the PPROP and PPS commands and the PLIST operation.

Examples: Put the properties like this:

```
?PPROP "FROG "I.D. "FREDDY  
?PPROP "FROG "COLOR "GREEN
```

Then, get the properties like this:

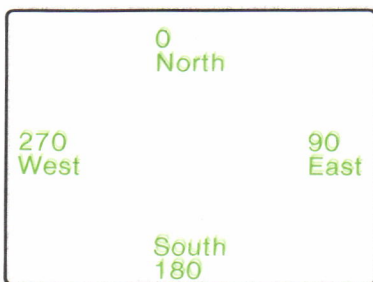
```
?SHOW GPROP "FROG "I.D.  
FREDDY  
?SHOW GPROP "FROG "COLOR  
GREEN  
  
?SHOW GPROP "ANYNAME "ANYPROP  
[]
```

HEADING

Operation

Format: HEADING

Remarks: Outputs the turtle's heading, a decimal number greater than or equal to 0 and less than 360. Logo follows the compass system, where North is a heading of 0 degrees, East 90, South 180, and West 270. When you start up Logo, the turtle has a heading of 0 (straight up).



Examples: `IF HEADING = 180 [PR [YOU ARE HEADED DUE SOUTH]]`

will print

`YOU ARE HEADED DUE SOUTH`

whenever the turtle is heading straight down.

HEADING

Operation

```
TO POLY :SIDE :ANGLE :START  
FORWARD :SIDE  
RIGHT :ANGLE  
IF :START = HEADING [STOP]  
POLY :SIDE :ANGLE :START  
END
```

```
?POLY 50 144 HEADING
```

POLY draws regular polygons. It stops when the turtle's current direction (HEADING) is the same as its starting direction (:START).

HIDETURTLE (HT)

Command

Format: HIDETURTLE

Remarks: Makes the turtle invisible. The turtle can still draw when it is hidden; in fact, it draws faster. If you have changed the turtle's shape with SETSHAPE, HIDETURTLE will hide that shape.



FD 50



HIDETURTLE



RT 90 FD 20

Example: Run the circle procedure with the turtle visible, then run it with the turtle invisible.

```
TO CIRCLE  
REPEAT 360 [FD 1 RT 1]  
END
```

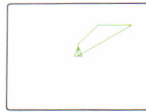
```
?CIRCLE  
?HIDETURTLE  
?CS  
?CIRCLE
```

Format: HOME

Remarks: Moves the turtle to the center of the screen and sets its heading to 0. This command is equivalent to SETPOS [0 0] SETHEADING 0. If the turtle's pen is down, the turtle leaves a trace.



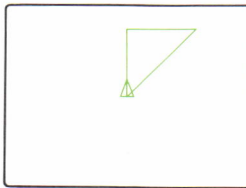
```
FD 15 RT 45
FD 25 RT 45
FD 35
```



HOME

Example: The following procedure draws a right-angled isosceles triangle with the turtle starting and ending at the center of the screen.

```
TO RTTRI :SIDE
FORWARD :SIDE
RIGHT 90
FORWARD :SIDE
HOME
END
```



```
?CS
?RTTRI 50
```


IF

Command or Operation

Format: IF *pred instructionlist1*
IF *pred instructionlist1 instructionlist2*

Remarks: If *pred* is TRUE then *instructionlist1* is run. If *pred* is FALSE, and there is a second instructionlist, then *instructionlist2* is run. If *pred* is FALSE, and there is no second instructionlist, then nothing is done.

In either case, if the selected instructionlist outputs anything, then IF outputs what instructionlist outputs. If the list does not output anything, neither does IF.

Examples: The procedure DECIDE is written in three equivalent ways. The first two use IF as a command: one version with two inputs to IF and one with three inputs. The third version of DECIDE uses IF (with three inputs) as an operation.

IF as a command:

```
TO DECIDE  
IF 0 = RANDOM 2 [OP "YES]  
OP "NO  
END
```

```
TO DECIDE  
IF 0 = RANDOM 2 [OP "YES] [OP "NO]  
END
```

IF Command or Operation

IF as an operation:

```
TO DECIDE  
OUTPUT IF 0 = RANDOM 2 ["YES] ["NO]  
END
```

You will get the answer YES or NO no matter which way you write the procedure DECIDE.

```
?PRINT DECIDE  
YES
```

IF can be used inside another IF clause. For example,

```
TO POSITIVE :NUM  
IF NUMBERP :NUM [IF :NUM > 0 [PR [POSIT→  
IVE NUMBER]] [PR [NEGATIVE NUMBER]]] [P→  
R [NOT A NUMBER]]  
END
```

IFFALSE (IFF) Command

Format: IFFALSE *instructionlist*

Remarks: Runs *instructionlist* if the result of the most recent TEST was FALSE; otherwise, IFFALSE does nothing. See the TEST and IFTRUE commands.

Example: TO CAPITAL.QUIZ
 PRINT [WHAT IS THE CAPITAL OF CANADA?]
 TEST READLIST = [OTTAWA]
 IFTRUE [PRINT "CORRECT!]
 IFFALSE [PRINT "INCORRECT]
 END

 ?CAPITAL.QUIZ
 WHAT IS THE CAPITAL OF CANADA?
 TORONTO
 INCORRECT

CAPITAL.QUIZ tests whether you know the capital of Canada. If you type the correct answer, Logo prints CORRECT!. If you type the wrong answer, Logo prints INCORRECT.

IFTRUE (IFT) Command

Format: IFTTRUE *instructionlist*

Remarks: Runs *instructionlist* if the result of the most recent TEST was TRUE; otherwise, IFTTRUE does nothing. See the TEST command.

Example: TO ADD.QUIZ
PRINT [HOW MUCH IS 5 + 7?]
TEST READWORD = 12
IFTRUE [PRINT [RIGHT!]] STOP]
PRINT [NO, TRY AGAIN]
ADD.QUIZ
END

?ADD.QUIZ
HOW MUCH IS 5 + 7?
11
NO, TRY AGAIN
HOW MUCH IS 5 + 7?
12
RIGHT!

ADD.QUIZ gives an arithmetic example. If you type the right answer, the procedure stops; otherwise, keep trying.

INT

Operation

Format: INT n

Remarks: Stands for “INTEger”.

Outputs the integer portion of n by removing the fractional portion, if any. If you want to round instead of truncate, then see the ROUND operation.

Examples: ?PRINT INT 5.2129
5

?PRINT INT 5.5129
5

?PRINT INT 5
5

?PRINT INT -5.8
-5

?PRINT INT -12.3
-12

The procedure INTP

- outputs TRUE if its input is an integer.

INT Operation

- outputs FALSE if its input is a real number with a fractional part.
- outputs "NOT A NUMBER" if its input is not a number.

```
TO INTP :N  
IF NOT NUMBERP :N [OUTPUT [NOT A NUMBER]]  
OUTPUT :N = INT :N  
END
```

```
?PRINT INTP 17  
TRUE
```

```
?PRINT INTP 100 / 8  
FALSE
```

```
?PRINT INTP "ONE  
NOT A NUMBER
```

```
?PRINT INTP SQRT 50  
FALSE
```

```
?PRINT INTP 30.0  
TRUE
```

The number 30.0 is an integer as far as Logo is concerned.

ITEM

Operation

Format: ITEM *n object*

Remarks: Outputs the *n*th element of *object* (a word or a list). If *n* is greater than the number of items in *object* or if the word or list is empty, Logo prints the error message:

TOO FEW ITEMS IN *object*

Example: ?MAKE "PETS [DOG CAT HAMSTER CANARY]
 ?PRINT ITEM 3 :PETS
 HAMSTER

 ?PRINT ITEM 1 :PETS
 DOG

 ?PRINT ITEM 5 :PETS
 TOO FEW ITEMS IN [DOG CAT HAMSTER CANAR→
 Y]

 ?PRINT ITEM 4 "COMPUTER
 P

The following procedure outputs a random item in a list.

```
TO RANPICK :GROUP
  OUTPUT ITEM (1 + RANDOM COUNT :GROUP) :>
  GROUP
END
```

```
?PRINT RANPICK [RED YELLOW BLUE GREEN]
BLUE
```


KEYP Operation

Format: KEYP

Remarks: Outputs TRUE if there is at least one character waiting to be read from the keyboard or current read device—one that has been typed on the keyboard but not yet read by READCHAR or READLIST. KEYP outputs FALSE if there are no unread characters.

If a file has been opened for reading, and if you have reached the end-of-file position, KEYP outputs FALSE.

Example: TO STEER
 FD 2
 IF KEYP [TURN READCHAR]
 STEER
 END

```
TO TURN :DIR
IF :DIR = "R [RT 10]
IF :DIR = "L [LT 10]
END
```

?STEER

The turtle moves continuously in one direction until you press R or L. In this way, you can make turtle drawings on the screen with the touch of a key.

LABEL

Command

Format: LABEL *name*

Remarks: Used only inside a procedure, LABEL gives the name to the instruction line to which the command GO *name* must pass control. See the GO command.

Example: The procedure TAKEOFF removes one letter each time the word is printed.

```
TO TAKEOFF :WD  
LABEL "LOOP  
PRINT :WD  
MAKE "WD BL :WD  
IF NOT EMPTY :WD [GO "LOOP]  
END
```

```
?TAKEOFF "IBM  
IBM  
IB  
I
```

LAST Operation

Format: **LAST** *object*

Remarks: Outputs the last element of *object*. If you try to print LAST of an empty word or an empty list, Logo prints the error message:

LAST DOESN'T LIKE *object* AS INPUT

Examples: ?SHOW LAST [JUDY SUSAN JOANNE]
 JOANNE

?SHOW LAST "VANILLA
A

?SHOW LAST [VANILLA]
VANILLA

?SHOW LAST [[THE A] FLAVOR IS [VANILLA →
CHOCOLATE STRAWBERRY]]
[VANILLA CHOCOLATE STRAWBERRY]

?PRINT LAST "
LAST DOESN'T LIKE AS INPUT

?PRINT LAST []
LAST DOESN'T LIKE [] AS INPUT

LAST Operation

The following procedure reverses a word or list and prints each element on a separate line.

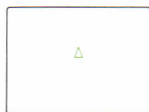
```
TO PRINTBACK :INPUT
IF EMPTY? :INPUT [STOP]
PRINT LAST :INPUT
PRINTBACK BUTLAST :INPUT
END
```

```
?PRINTBACK "LOGO
O
G
O
L
```

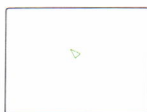
LEFT (LT) Command

Format: LEFT *degrees*

Remarks: Turns the turtle left (counterclockwise) the specified number of *degrees*. The *degrees* can be any real number from -9999 through 9999.



CS

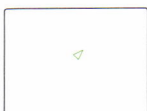


LEFT 45

LEFT 45 turns the turtle 45 degrees left.



CS



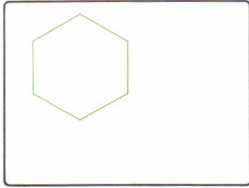
LEFT -45

LEFT -45 turns the turtle 45 degrees right.

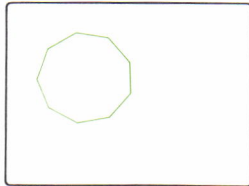
Examples: The procedure POLY draws figures like those illustrated.

```
TO POLY :SIDE :ANGLE  
FORWARD :SIDE  
LEFT :ANGLE  
POLY :SIDE :ANGLE  
END
```

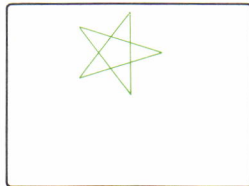
LEFT (LT) Command



?CS
?POLY 70 60



?CS
?POLY 30 40



?CS
?POLY 80 144

To stop the procedure, press Ctrl-Break.

LIST Operation

Format: LIST *object1 object2*
 (LIST *object1 object2 object3 . . .*)
 (LIST *object1*)

Remarks: Outputs a list whose elements are *object1*, *object2*,
 etc. The object can be a word or a list.

If LIST has one input, parentheses are needed only if there is something else on the line following the LIST and its input:

```
?SHOW (LIST "TOADS) SHOW "TOADS  
[TOADS]  
TOADS
```

If there are more than two inputs to LIST, you need parentheses around LIST and its inputs.

```
?SHOW (LIST "ALPHA "BETA "GAMMA)  
[ALPHA BETA GAMMA]
```

LIST

Operation

Examples: ?SHOW LIST "ROSE [TULIP CHRYSANTHEMUM]
[ROSE [TULIP CHRYSANTHEMUM]]

?SHOW (LIST "ROSE "TULIP "CHRYSANTHEMUM→
)
[ROSE TULIP CHRYSANTHEMUM]

?SHOW LIST [A QUICK BROWN FOX] [LOOKS AT→
THE LAZY FROG]
[[A QUICK BROWN FOX] [LOOKS AT THE LAZY→
FROG]]

?SHOW LIST "A []
[A []]

?MAKE "ANIMALS "TOADS
?SHOW LIST :ANIMALS
[TOADS]
?SHOW (LIST :ANIMALS) PR "RELAX
[TOADS]
RELAX

LIST Operation

The following chart compares four primitives that combine words and lists:

Operation	Input 1	Input 2	Output
FPUT	"COW	"HORSE	error
LIST	"COW	"HORSE	[COW HORSE]
LPUT	"COW	"HORSE	error
SENTENCE	"COW	"HORSE	[COW HORSE]
FPUT	"LOGO	[IS WONDERFUL]	[LOGO IS WONDERFUL]
LIST	"LOGO	[IS WONDERFUL]	[LOGO [IS WONDERFUL]]
LPUT	"LOGO	[IS WONDERFUL]	[IS WONDERFUL LOGO]
SENTENCE	"LOGO	[IS WONDERFUL]	[LOGO IS WONDERFUL]
FPUT	[THE FOX]	[LOOKS AT FIDO]	[[THE FOX] LOOKS AT FIDO]
LIST	[THE FOX]	[LOOKS AT FIDO]	[[THE FOX] [LOOKS AT FIDO]]
LPUT	[THE FOX]	[LOOKS AT FIDO]	[LOOKS AT FIDO [THE FOX]]
SENTENCE	[THE FOX]	[LOOKS AT FIDO]	[THE FOX LOOKS AT FIDO]
FPUT	"COMPUTERS	[]	[COMPUTERS]
LIST	"COMPUTERS	[]	[COMPUTERS []]
LPUT	"COMPUTERS	[]	[COMPUTERS]
SENTENCE	"COMPUTERS	[]	[COMPUTERS]

LISTP

Operation

Format: LISTP *object*

Remarks: Outputs TRUE if *object* is a list; otherwise, FALSE.

Examples: ?PRINT LISTP 3
FALSE

```
?PRINT LISTP [3]  
TRUE
```

```
?PRINT LISTP [ ]  
TRUE
```

```
?PRINT LISTP "  
FALSE
```

```
?PRINT LISTP BUTFIRST "CHOCOLATE  
FALSE
```

```
?PRINT LISTP BUTFIRST [CHOCOLATE]  
TRUE
```

The following procedure determines whether its input is a word or a list.

```
TO CHECKLIST :OBJ  
IF LISTP :OBJ [PR [THIS IS A LIST]] [PR→  
[THIS IS A WORD]]  
END
```

```
?CHECKLIST "F  
THIS IS A WORD
```

LN Operation

Format: $\text{LN } a$

Remarks: Outputs the natural logarithm of the number a (that is, the logarithm in base e). The result is rounded to PRECISION significant digits, where PRECISION must be less than or equal to 100.

The a must be a positive number.

When PRECISION is greater than 100, Logo prints the error message:

LN DOESN'T LIKE PRECISION n

When a is 0 or negative, Logo prints the error message:

LN DOESN'T LIKE a AS INPUT

Note that LN is the inverse of the operation EXP.

LN Operation

Examples: ?SETPRECISION 5

?PRINT LN 2

0.69315

?PRINT LN -55

LN DOESN'T LIKE -55 AS INPUT

?PRINT LN 1

0

?SETPRECISION 20

?PRINT LN 10

2.302585092994045684

?SETPRECISION 150

?PRINT LN 10

LN DOESN'T LIKE PRECISION 150

?SETPRECISION 10

TO LOG2 :N

OUTPUT (LN :N) / LN 2

END

?PRINT LOG2 1024

10

LOG2 shows how to obtain a logarithm in any base.
Here we've used base 2.

Comments: The operations LN and EXP are *inverse* to each other. For example:

```
?SETPRECISION 10
?PRINT EXP LN 3.17777777
3.17777776
```

```
?PRINT LN EXP 3.17777777
3.17777777
```

The last two outputs are to be compared with the value of the inputs.

More generally, you may numerically check the validity of mathematical identities such as

$$e^{\log x} = x$$

$$\log e^x = x$$

$$\log(xy) = \log x + \log y$$

This can be done by comparing for various choices of :X and :Y and various precisions:

```
EXP LN :X with :X
LN EXP :X with :X
LN :X *:Y with (LN :X) + (LN :Y)
```

Because these identities involve more than one operation in succession, roundoff errors may occur. Accordingly, there may be a slight difference in the last digits of the outputs when they are compared.

LOAD Command

Format: `LOAD filespec`
 `LOAD filespec package`

Remarks: Loads the contents of *filespec* from disk. The *filespec* is any valid existing file name, as explained under “Naming Files”, in Chapter 4.

If no extension is specified in *filespec*, *file name.LF* is loaded. If a file extension is specified, *file name.extension* is loaded. To load a file having no extension, a period is needed after the file name in order to override the automatic .LF extension.

All variables and procedures go back into the packages they were in when they were saved, unless LOAD has a second input.

If there is a second input, it specifies the *package* into which everything is loaded, regardless of the package from which it was saved. See the PACKAGE command. If anything in the file is in a buried package, it is buried after being loaded into the workspace.

If *filespec* doesn't exist, the following error message is printed:

FILE *filespec* DOES NOT EXIST

Note: If you want to write a procedure which loads a program file into the workspace in order to be executed immediately, see “Comments” in COPYDEF command.

Examples: `?LOAD "BEAR`

Everything in the file named BEAR.LF is read into the workspace. Each variable or procedure is in the package it came from.

`LOAD "BEAR.`

Everything in the file named BEAR is read into the workspace.

`?LOAD "BEAR.GRA "SHAPES`

Everything in the file named BEAR.GRA is read into the workspace in the package SHAPES.

LOADPIC

Command

Format: `LOADPIC filespec`

Remarks: The *filespec* is a file name that was used by SAVEPIC. Loads the picture named by *file name*.PIC, if no extension has been specified, onto the graphics or text screen.

You must have saved the picture by using SAVEPIC in order to use this command. Because SAVEPIC saves the screen video buffer into a file, the picture can be either graphics or text.

If you have an IBM Monochrome Display, LOADPIC loads the file onto the textscreen. If you have a TV or monitor, LOADPIC loads the file onto the graphics screen. See “Screen File” in Chapter 4 and the SAVEPIC command for more information.

Examples: `?LOADPIC "CAT`

Loads the screen contained in the file CAT.PIC onto the graphics screen.

`?LOADPIC "CAT.`

Loads the screen contained in the file CAT onto the screen.

LOCAL Command

Format: LOCAL *name*
 (LOCAL *name1 name2 ...*)

Remarks: Makes one or more names local (specific) to the procedure within which LOCAL and MAKE are used. This means that each *name* is accessible only to that procedure and to procedures it calls. In this regard, they resemble inputs to the procedure.

LOCAL can be used only inside a procedure. If you try to use LOCAL outside a procedure, Logo prints the error message:

CAN BE USED ONLY IN A PROCEDURE

If LOCAL has more than one input, you must put LOCAL and its inputs within parentheses.

Examples: TO YESNO :QUESTION
 LOCAL "ANSWER
 PR :QUESTION
 MAKE "ANSWER READWORD
 IF EQUALP :ANSWER "YES [OUTPUT "TRUE]
 OUTPUT "FALSE
 END

 TO SPORTS
 PR [NAME A SPORT]
 MAKE "ANSWER READLIST
 IF YESNO [DO YOU PLAY THIS SPORT?] [PR →
 [THAT'S GOOD]] [PR [TOO BAD]]
 PR SENTENCE :ANSWER [IS A FUN SPORT]
 END

LOCAL Command

```
?SPORTS  
NAME A SPORT  
SCUBA DIVING  
DO YOU PLAY THIS SPORT?  
NO  
TOO BAD  
SCUBA DIVING IS A FUN SPORT
```

SPORTS asks you to name a sport and **YESNO** checks if you play the sport.

Imagine what happens if the **LOCAL** is omitted from **YESNO**. Each procedure uses a variable named **ANSWER** to hold your answer to a question. Because the variables are not local, the procedure **YESNO** destroys the value that **SPORTS** expects to have in that variable.

```
?SPORTS  
NAME A SPORT  
SCUBA DIVING  
DO YOU PLAY THIS SPORT?  
NO  
TOO BAD  
NO IS A FUN SPORT
```

Format: LPUT *object list*

Remarks: Stands for “Last PUT”.

Outputs a new list formed by putting *object* at the end of *list*. See the LIST command for a chart comparing LPUT with other primitives that combine words and lists.

Examples: ?SHOW LPUT "GERBIL [HAMSTER GUINEA.PIG]
[HAMSTER GUINEA.PIG GERBIL]

?SHOW LPUT [THE A AN] [CAT ELEPHANT]
[CAT ELEPHANT [THE A AN]]

?SHOW LAST LPUT "GERBIL [HAMSTER GUINEA→
.PIG]
GERBIL

?SHOW LPUT "S "FRIEND
LPUT DOESN'T LIKE FRIEND AS INPUT

“FRIEND is not a list.

LPUT

Operation

The following procedure adds a new entry to an English-Spanish dictionary.

```
TO NEWENTRY :ENTRY
MAKE "DICTIONARY LPUT :ENTRY :DICTIONAR→
Y
END
```

```
?MAKE "DICTIONARY [[HOUSE CASA] [SPANIS→
H ESPANOL] [HOW COMO]]
?SHOW :DICTIONARY
[[HOUSE CASA] [SPANISH ESPANOL] [HOW CO→
MO]]
?NEWENTRY [TABLE MESA]
?SHOW :DICTIONARY
[[HOUSE CASA] [SPANISH ESPANOL] [HOW CO→
MO] [TABLE MESA]]
```

MAKE Command

Format: MAKE *name object*

Remarks: Gives the value *object* to the variable *name*. After the variable is created, you can access the contents by using *:name*. The : (colon) means “the thing that is called”. See the LOCAL command to keep *name* local to the procedure in which MAKE *name* is used.

Examples: ?MAKE "JOB 259
 ?PRINT :JOB
 259

 ?MAKE "JOB "WELDER
 ?PRINT :JOB
 WELDER

 ?MAKE "WELDER 32
 ?PRINT :WELDER
 32

 ?PRINT THING :JOB
 32

See the THING operation for details.

 ?MAKE :JOB [JENNY SPARROW]

MAKE Command

At this point, :JOB is WELDER and THING :JOB is [JENNY SPARROW].

```
?PRINT "JOB
JOB
```

```
?PRINT :JOB
WELDER
```

```
?PRINT THING "JOB
WELDER
```

```
?PRINT THING :JOB
JENNY SPARROW
```

The following procedure prints a comment on today's weather:

```
TO WEATHER
PR [WHAT'S THE WEATHER LIKE TODAY?]
MAKE "ANSWER READLIST
IF :ANSWER = [RAINING] [PR [I WISH IT W→
OULD STOP RAINING] STOP]
IF :ANSWER = [SUNNY] [PR [I HOPE IT STA→
YS SUNNY] STOP]
PR (SE [I WONDER IF IT WILL BE] :ANSWER→
  "TOMMORROW)
END
```

MAKE Command

?WEATHER
WHAT'S THE WEATHER LIKE TODAY?
SUNNY
I HOPE IT STAYS SUNNY

?WEATHER
WHAT'S THE WEATHER LIKE TODAY?
CLOUDY
I WONDER IF IT WILL BE CLOUDY TOMORROW

?WEATHER
WHAT'S THE WEATHER LIKE TODAY?
RAINING
I WISH IT WOULD STOP RAINING

MEMBERP

Operation

Format: MEMBERP *object1 object2*

Remarks: Outputs TRUE if *object1* (a character, word, or list) is an element of *object2*; otherwise outputs FALSE.

Note: A character only can be an element of a word, a word an element of a list, and a list an element of a list of lists.

The empty word is not a member of the empty list.

```
?PRINT MEMBERP " [ ]  
FALSE
```

It is possible, however, to construct a list which contains the empty word.

```
?MAKE "NULL LIST "  
?PRINT MEMBERP " :NULL  
TRUE
```

Examples: ?PRINT MEMBERP 3 [2 5 3 6]
TRUE

```
?PRINT MEMBERP 3 [2 5 [3] 6]  
FALSE
```

```
?PRINT MEMBERP [2 5] [2 5 3 6]  
FALSE
```

```
?PRINT MEMBERP "B "RABBIT  
TRUE
```

```
?PRINT MEMBERP "FLORIDA [CALIFORNIA FLO →  
RIDA IOWA]  
TRUE
```


MEMBERP Operation

```
?PRINT MEMBERP [FLORIDA GEORGIA] [[FLOR →  
IDA GEORGIA] IOWA]  
TRUE
```

```
?PRINT MEMBERP [FLORIDA GEORGIA] [FLORI →  
DA GEORGIA]  
FALSE
```

A list cannot be a member of itself.

The following procedure determines whether its input is a vowel.

```
TO VOWELP :LETTER  
  OUTPUT MEMBERP :LETTER [A E I O U]  
END
```

```
?PRINT VOWELP "F  
FALSE  
?PRINT VOWELP "A  
TRUE
```

MIXEDSCREEN (MS)

Command

Format: MIXEDSCREEN

Remarks: Makes the screen available for both graphics and text.

The first graphics command given after you start up Logo will automatically switch you to mixedscreen mode.

Initially, the whole screen is available for graphics, and the bottom six lines can be used for text.

You can change the portion of the screen available for text by using the SETTEXT command. MIXEDSCREEN uses the current number of lines from SETTEXT to determine how many lines of text are available on the graphics screen.

The MIXEDSCREEN command is equivalent to pressing the F2 function key. The F2 key will have no effect if a graphics command has not yet been used after starting Logo, or if the cursor is in the graphics portion of the screen.

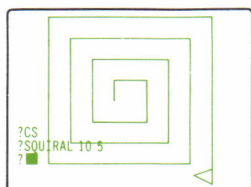
If you are in textscreen mode and use a graphics command, you will automatically return to either fullscreen or mixedscreen mode (whichever you used last).

However if .SCREEN is 2, neither MIXEDSCREEN nor the F2 function key will have any effect.

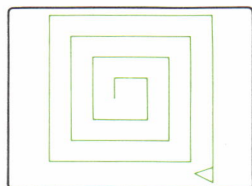
MIXEDSCREEN (MS)

Command

Examples: TO SQUIRAL :STEP :INC
 IF :STEP > 80 [STOP]
 FD :STEP
 RT 90
 SQUIRAL :STEP + :INC :INC
 END



?MIXEDSCREEN
 ?CS
 ?SQUIRAL 10 5



?FULLSCREEN

NAME Command

Format: NAME *object name*

Remarks: Makes *object* the value of *name*. NAME is equivalent to MAKE with the order of the inputs reversed. See the MAKE command.

Examples: ?NAME 259 "JOB
?PRINT :JOB
259
?NAME "BRICKLAYER "JOB
?PRINT :JOB
BRICKLAYER

However,

```
?PRINT :BRICKLAYER  
BRICKLAYER HAS NO VALUE
```

Note that

```
NAME "BRICKLAYER "JOB
```

has the same effect as

```
MAKE "JOB "BRICKLAYER
```

NAME Command

The following procedure uses NAME to remember your name in order to greet you.

```
TO GREET  
  PRINT [HELLO, WHAT'S YOUR NAME?]  
  NAME READWORD "ANSWER  
  (PRINT [HI] :ANSWER)  
END
```

```
?GREET  
HELLO, WHAT'S YOUR NAME?  
GINNY  
HI GINNY
```

NAMEP

Operation

Format: NAMEP *name*

Remarks: Outputs TRUE if *name* has a value, that is, if *:name* exists; otherwise FALSE.

Examples: ?PRINT :ANIMAL
ANIMAL HAS NO VALUE

```
?PRINT NAMEP "ANIMAL  
FALSE
```

```
?MAKE "ANIMAL "AARDVARK  
?PRINT NAMEP "ANIMAL  
TRUE
```

```
?PRINT :ANIMAL  
AARDVARK
```

The following procedure tests whether :COUNTRY has a value, and prints the value, if any.

```
TO CAPITAL :COUNTRY  
TEST NAMEP :COUNTRY  
IFFALSE [PRINT [I DON'T KNOW]]  
IFTRUE [PRINT (SE THING :COUNTRY [IS TH→  
E CAPITAL OF] :COUNTRY)  
END
```

```
?MAKE "SENEGAL "DAKAR  
?CAPITAL "SENEGAL  
DAKAR IS THE CAPITAL OF SENEGAL  
?CAPITAL "GREECE  
I DON'T KNOW
```

NODES Operation

Format: NODES

Remarks: Outputs the number of free nodes. This gives you an idea of how much space you have in your workspace for procedures, variables, and the running of procedures. If you want to find out how many nodes you have left, run NODES immediately after RECYCLE.

NODRIBBLE

Command

Format: NODRIBBLE

Remarks: Turns off the dribble feature so a copy of the characters from the screen will no longer be sent to the device or file used previously by the command DRIBBLE. For more information on dribble files, see the DRIBBLE command and “Dribble File” in Chapter 4.

Examples: ?DRIBBLE "B:NUMBERS

creates a file called NUMBERS on a diskette in drive B and starts the dribbling process.

```
?REPEAT 5 [PRINT RANDOM 10]
8
0
3
3
2
?NODRIBBLE
```

Everything put on the screen after the DRIBBLE line is also sent to the file NUMBERS, until you stop the dribble feature. Now, if you print out the file NUMBERS, it contains what you just typed.

```
?PROFILE "B:NUMBERS
?REPEAT 5 [PRINT RANDOM 10]
8
0
3
3
2
?NODRIBBLE
```


NOT Operation

Format: NOT *pred*

Remarks: Outputs TRUE if *pred* is FALSE; outputs FALSE if *pred* is TRUE.

Examples: ?PRINT NOT EQUALP "A "B
TRUE

?PRINT NOT EQUALP "A "A
FALSE

?PRINT NOT "A = FIRST "DOG
TRUE

?PRINT NOT "A
A IS NOT TRUE OR FALSE

If **WORDP** were not a primitive, it could be defined as follows:

TO WORD? :OBJ
OUTPUT NOT LISTP :OBJ
END

?PRINT WORD? [LOOK]
FALSE

?PRINT WORD? "LOOK
TRUE

NOT Operation

The following procedure tells whether its input is a “word that isn’t a number”.

```
TO REALWORDP :OBJ  
OP AND WORDP :OBJ NOT NUMBERP :OBJ  
END
```

```
?PRINT REALWORDP 22.5  
FALSE
```

```
?PRINT REALWORDP "KANGAROO  
TRUE
```

```
?PRINT REALWORDP FIRST PEN  
TRUE
```

PEN is the list describing the current state of the turtle’s pen. See the PEN operation.

NUMBERP Operation

Format: NUMBERP *object*

Remarks: Outputs TRUE if *object* is a number; otherwise, FALSE.

Examples: ?PRINT NUMBERP 3
TRUE

```
?PRINT NUMBERP [3]  
FALSE
```

```
?PRINT NUMBERP [ ]  
FALSE
```

```
?PRINT NUMBERP "  
FALSE
```

```
?PRINT NUMBERP BUTFIRST 3165.2  
TRUE
```

```
?PRINT NUMBERP BUTFIRST [ELEPHANT]  
FALSE
```

The following procedure tells whether its input is a list of numbers.

```
TO NUMLISTP :LIST  
IF :LIST = [ ] [OP "FALSE]  
IF COUNT :LIST = 1 [OP NUMBERP FIRST :L →  
IST]  
OP AND NUMBERP FIRST :LIST NUMLISTP BF →  
:LIST  
END
```

```
?PRINT NUMLISTP [1 2 3]  
TRUE
```

OPEN

Command

Format: OPEN *device*
 OPEN *filespec*

Remarks: The *device* is any valid device and *filespec* is any valid file as described under “Naming Files” in Chapter 4. OPEN opens the *device* or *filespec* so it can send or receive characters. OPEN must be used before you can use any primitive that accesses a data file. See Chapter 4, “File Handling” for more information.

You can open a maximum of five devices or files at once. If the file named by *filespec* does not exist, then OPEN creates a file with this name. All devices or files that are open must be closed. See the CLOSE and CLOSEALL commands to close files and devices.

Example: TO READFILE :FILE
 SETREAD :FILE
 IF READEOFP [CLOSE :FILE STOP]
 PRINT READLIST
 READFILE :FILE
 END

```
?OPEN "ADDRESS.DAT
?READFILE "ADDRESS.DAT
ADDRESS LIST
MARIE: 55 CEDARWOOD
LOGO: 9960 COTE DE LIESSE
```

The READFILE procedure reads information from a file that is already open until the end-of-file position is reached. At that time, the file is closed and execution of the procedure stops.

OR Operation

Format: *OR pred1 pred2*
 (OR pred1 pred2 pred3 ...)
 (OR pred1)

Remarks: Outputs FALSE if all of its inputs are false; otherwise, TRUE. Its inputs must be TRUE or FALSE. If OR has one input or more than two inputs, OR and its inputs must be enclosed in parentheses.

Examples: ?PRINT OR "TRUE "TRUE
 TRUE

 ?PRINT OR "TRUE "FALSE
 TRUE

 ?PRINT OR "FALSE "FALSE
 FALSE

 ?PRINT (OR "FALSE "FALSE "FALSE "TRUE)
 TRUE

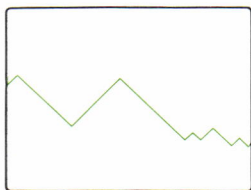
 ?PRINT OR 5 7
 5 IS NOT TRUE OR FALSE

OR Operation

The procedure MOUNTAINS draws mountains.

```
TO MOUNTAINS  
  SETPC 1 HT  
  RT 45  
  FD 5  
  SUBMOUNTAIN  
END
```

```
TO SUBMOUNTAIN  
  FD 5 + RANDOM 10  
  IF OR YCOR > 50 YCOR < 0 [SETHEADING 18 →  
    0 - HEADING]  
  SUBMOUNTAIN  
END
```



?MOUNTAINS

Press Ctrl-Break to stop.

OUTPUT (OP) Command

Format: OUTPUT *object*

Remarks: This command is meaningful only when it is within a procedure, not at top level. It makes *object* the output of this procedure and returns control to the caller. Note that OUTPUT is itself a command, but the procedure containing it is an operation because the procedure is made to output. Compare with the STOP command.

Examples: TO MARK.TWAIN
OUTPUT [SAMUEL CLEMENS]
END

```
?PR SE MARK.TWAIN [IS A GREAT AUTHOR]
SAMUEL CLEMENS IS A GREAT AUTHOR
```

The procedure WHICH outputs the position of an element in a list.

```
TO WHICH :MEMBER :LIST
IF NOT MEMBERP :MEMBER :LIST [OP 0]
IF :MEMBER = FIRST :LIST [OP 1]
OP 1 + WHICH :MEMBER BF :LIST
END
```

```
?MAKE "VOWELS [A E I O U]
?PR WHICH "E :VOWELS
2
```

```
?PR WHICH "U :VOWELS
5
```

```
?PR WHICH "W :VOWELS
0
```

OUTPUT (OP)

Command

Here is a definition of the absolute-value operation:

```
TO ABS :N
IF :N < 0 [OUTPUT -:N] [OUTPUT :N]
END
```

```
?PR ABS 5
5
```

```
?PR ABS -5
5
```

The following operation tells whether its first input (a character) is an element of its second input (a word). The primitive **MEMBERP** does this for words and lists.

```
TO INP :CHAR :WRD
IF EMPTY? :WRD [OUTPUT "FALSE]
IF EQUALP :CHAR FIRST :WRD [OUTPUT "TRUE]
E]
OUTPUT INP :CHAR BUTFIRST :WRD
END
```

```
?PRINT INP "A "TEACUP
TRUE
```

```
?PRINT INP "TEA "TEACUP
FALSE
```

```
?PRINT INP "I "SAUCER
FALSE
```


PACKAGE Command

Format: `PACKAGE` *package name*
 `PACKAGE` *package namelist*

Remarks: Puts each named procedure in *package*. It does so by putting the `PROCPKG` property with value *package* on the property list associated with the *name* of each procedure. `PROCPKG` is a special word, which is always given as a property to any procedure in a package.

This primitive is very useful for organizing your procedures in separate files. See “Organizing Your Procedures in Files” in Chapter 4 for more information.

PACKAGE

Command

Examples: `PACKAGE "SHAPES [TRIANGLE SQUARE]`

puts TRIANGLE and SQUARE in the package SHAPES.

```
?POTS "SHAPES  
TO TRIANGLE  
TO SQUARE
```

prints out the procedure titles in the package SHAPES.

```
?SAVE "GORDON "SHAPES
```

saves the procedures in the package SHAPES in the diskette file called GORDON.

```
?PPS "SHAPES  
SQUARE'S PROCPKG IS SHAPES  
TRIANGLE'S PROCPKG IS SHAPES
```

prints the properties of everything in the SHAPES package. Note that the PROCPKG property with the value SHAPES is associated with TRIANGLE and SQUARE.

PADDLE Operation

Format: PADDLE *paddlenumber*

Remarks: Outputs an integer, usually between 4 and 130, representing the rotation of the dial on the specified paddle. This number varies according to the minimal and maximal resistance of the dial on the paddle being used. The *paddlenumber* must be 0, 1, 2, or 3 because there are four paddles.

If you do not have a Game Control Adapter in your computer or have not plugged in your paddles, Logo prints the error message:

DEVICE UNAVAILABLE

PADDLE also is used for joysticks. You can have only two joysticks. PADDLE 0 and PADDLE 1 refer to the exact position of joystick 0, while PADDLE 2 and PADDLE 3 refer to joystick 1.

Example: The following procedure allows you to draw on the screen by rotating the paddle.

```
TO PDRAW  
  RIGHT (PADDLE 0) - 65  
  FORWARD PADDLE 1  
PDRAW  
END
```

```
?PDRAW
```

PALETTE (PAL)

Operation

Format: PALETTE

Remarks: Outputs the number of the current PALETTE (see SETPAL). If the palette is set to 1, the turtle's pen color can be cyan (1), magenta (2), or white (3). If the palette is set to 0, the pen color can be green (1), red (2), or brown (3).

When Logo starts, the palette is set to 1.

Examples: ?PRINT PALETTE
 1
 ?SETPC 2

The turtle's pen is set to 2, on palette 1. The turtle's pen color is magenta.

 ?SETPAL 0

When the palette number is set to 0, the turtle's pen color is red.

 ?PRINT PALETTE
 0

The palette number is 0.

PAUSE

Command or Operation

Format: PAUSE

Remarks: This command is meaningful only when it is within a procedure, not at top level. It suspends running of the procedure and tells you that you are pausing. You then can type instructions interactively.

To indicate that you are in a pause and not at top level, the prompt changes to the name of a procedure followed by a question mark.

During a pause, Ctrl-Break does not return you to top level. However, Ctrl-Break may be used to interrupt any procedure that you run within the pause. The only way to return to top level during a pause is to run THROW "TOPLEVEL.

All local variables are accessible during a pause (see PRINT :MAX in the example).

The procedure may be resumed by typing CO (except if you use PAUSE after using the command LOAD). If CO has an input, then PAUSE is an operation. CO's input becomes PAUSE's output.

Pressing the F5 function key will also cause a pause. It is possible to have a pause within a pause.

PAUSE

Command or Operation

Example: The following procedure makes the turtle move randomly around the screen.

```
TO WALK :MAX
RT RANDOM 360
FD RANDOM :MAX
PR POS
PAUSE
WALK :MAX
END

?WALK 100
60.4109 -13.947
PAUSING ... IN WALK:
PAUSE
WALK ?PRINT HEADING
103
WALK ?PRINT :MAX
100
WALK ?CO
68.4381 2.1059
PAUSING... IN WALK:
PAUSE
WALK ?THROW "TOplevel
?
```

PEN Operation

Format: PEN

Remarks: Outputs a three-word list describing the current state of the turtle's pen.

The first word is PENDOWN, PENERASE, PENUP, or PENREVERSE. (See the individual commands for further information.)

The second word is the number indicating the current pen color.

The third word is the number representing the current palette number.

When the turtle first starts up, PEN outputs

[PENDOWN 3 1]

The form of the output is suitable for input to SETPEN. See the SETPEN command for an example.

Examples: ?PENUP
?SETPAL 0
?SETPC 2
?PRINT PEN
PENUP 2 0

PENCOLOR (PC)

Operation

Format: PENCOLOR

Remarks: Outputs a number representing the current color of the pen.

Color	Palette 0	Palette 1
1	Green	Cyan
2	Red	Magenta
3	Brown	White

When the turtle first starts up, PENCOLOR is 3, and the palette number is 1. Changing the palette will not change the pen color number, although the actual color will change. See the PALETTE operation.

Note that 0 is the current background color.

Examples: ?SETPC 2
?PRINT PENCOLOR
2
?FORWARD 30

If the palette is 1, the pen color is magenta.

?SETPAL 0
?PRINT PENCOLOR
2
?FORWARD 30

The pen color is now red.

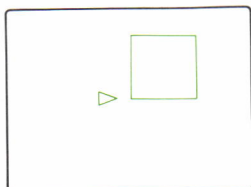
PENDOWN (PD)

Command

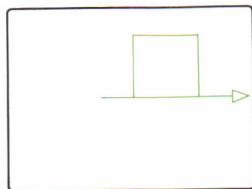
Format: PENDOWN

Remarks: Puts the turtle's pen down. When the turtle moves, it draws lines in the current pen color. The turtle begins with its pen down. If you have set the pen state to PENUP, then PENDOWN will allow the turtle to draw again.

Examples:



```
?REPEAT 4 [FD 50 RT 90]  
?RT 90  
?PENUP  
?BK 20
```



```
?PENDOWN  
?FD 90
```

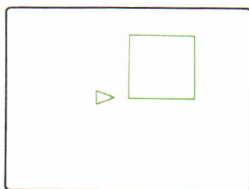
PENERASE (PE)

Command

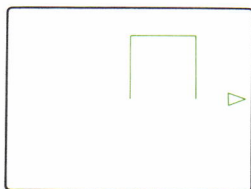
Format: PENERASE

Remarks: Puts the turtle's eraser down. When the turtle moves, it will erase any previously drawn line it passes over. To lift the eraser, you must use either PENDOWN or PENUP.

Examples:



```
?REPEAT 4 [FD 50 RT 90]  
?RT 90  
?PENUP  
?BK 20
```



```
?PENERASE  
?FD 90
```

PENREVERSE (PX)

Command

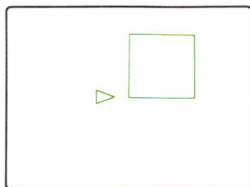
Format: PENREVERSE

Remarks: Puts the “reversing pen” down. When the turtle moves, it exchanges the pen color and background color, drawing where there aren’t lines and erasing where there are.

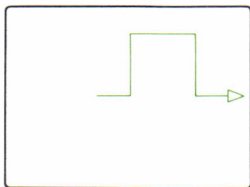
The exact effect of this reversal is complex; what it looks like on the screen depends on the pen color, background color, and whether lines are horizontal or vertical. The best results are seen on a black background.

To allow the turtle to draw normally again, use either PENUP or PENDOWN.

Examples:



```
?REPEAT 4 [FD 50 RT 90]  
?RT 90  
?PENUP  
?BK 20
```



```
?PENREVERSE  
?FD 90
```

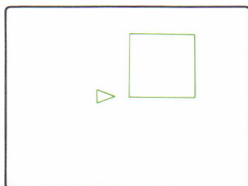
PENUP (PU)

Command

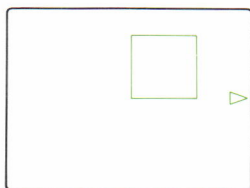
Format: PENUP

Remarks: Lifts the turtle's pen up. When the turtle moves, it does not draw lines. The turtle begins with its pen down. Once you have set the pen state to PENUP, PENDOWN allows the turtle to draw again.

Examples:



```
?REPEAT 4 [FD 50 RT 90]  
?RT 90  
?PENUP  
?BK 20
```



```
?FD 90
```

PI Operation

Format: PI

Remarks: Outputs the mathematical constant PI (the ratio of a circle's circumference to its diameter) rounded to PRECISION significant digits, where PRECISION must be less than or equal to 100.

If PRECISION is greater than 100, Logo displays the error message:

PI DOESN'T LIKE PRECISION *n*

The constant PI is useful, for example, to translate angles from degrees to radians and vice versa. More precisely:

:N degrees equals $(PI * :N) / 180$ radians

:M radians equals $(180 * :M) / PI$ degrees

PI

Operation

Examples: ?SETPRECISION 5

?PRINT PI

3.1416

?SETPRECISION 100

?PRINT PI

3.1415926535897932384626433832795028841→

971693993751058209749445923078164062862→

08998628034825342117068

?SETPRECISION 150

?PRINT PI

PI DOESN'T LIKE PRECISION 150

?SETPRECISION 5

?MAKE "N 45

?PRINT (PI * :N)/180

0.78539

?MAKE "M 2

?PRINT (180 * :M)/PI

114.59

PKGALL Command

Format: PKGALL *package*

Remarks: Puts into *package* all procedures and variables (in the workspace) that are not already in packages. These procedures then have the PROCPKG property with the value *package* on their property lists. The variables have the VALPKG property with the value *package* on their property lists.

VALPKG is a special word, which is always given as a property to any variable in a package. To package variables, you can use the command PKGALL or PPROP.

See the PACKAGE command for further information on PROCPKG and packaging.

PKGALL

Command

Examples: ?POTS

```
TO GREET  
TO LENGTH :OBJ
```

prints out the procedure titles.

```
?PONS  
F IS 3  
LIST IS [A B C]
```

prints out the names.

```
?PKGALL "LEFTOVER
```

puts everything in a *package* called LEFTOVER.

```
?PPS "LEFTOVER  
GREET'S PROCPKG IS LEFTOVER  
LENGTH'S PROCPKG IS LEFTOVER  
F'S VALPKG IS LEFTOVER  
LIST'S VALPKG IS LEFTOVER
```

prints the properties for the *package* LEFTOVER.
Note that the PROCPKG property is associated with
the procedures GREET and LENGTH. The
VALPKG property is associated with the variables F
and LIST.

PLIST Operation

Format: `PLIST name`

Remarks: Outputs the property list associated with *name*. This is a list of property names, paired with their values, in the form `PROP1 VAL1 PROP2 VAL2 ...`

Examples: `?SHOW PLIST "FROG`
 `[I.D. FREDDY COLOR GREEN LEGS 4]`

shows the property list associated with FROG.

`?SHOW PLIST ".SYSTEM`
`[BURY TRUE]`

shows the property list associated with .SYSTEM.
.SYSTEM is a special word that is the name of the package containing the system variables ERRACT and REDEFP.

`?SHOW PLIST "GREET`
`[PROCPKG LEFTOVER]`

Shows the property list associated with GREET.
GREET is a procedure in the package called LEFTOVER.

PO

Command

Format: PO *name*
PO *namelist*

Remarks: PO stands for "Print Out".

Prints the definitions of the named procedures.

Examples: ?PO "LENGTH
TO LENGTH :OBJ
IF EMPTY :OBJ [OP 0] [OP 1 + LENGTH BF→
:OBJ]
END

?PO [LENGTH GREET]
TO LENGTH :OBJ
IF EMPTY :OBJ [OP 0] [OP 1 + LENGTH BF→
:OBJ]
END

TO GREET
PR [GOOD MORNING. HOW ARE YOU TODAY?]
END

POALL Command

Format: POALL
POALL *package*
POALL *packagelist*

Remarks: When POALL has no input, it prints the definition of every procedure and the value of every variable from the workspace except the ones that are buried. See the BURY command.

If an input *package* or *packagelist* is present, POALL prints only the definitions of procedures and values of variables in that *package* or *packagelist* (including those that are buried in the *package*).

Because POALL takes an optional input, you cannot put another command immediately after POALL on the same line.

POALL

Command

Examples: ?POALL

```
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END
```

```
TO LENGTH :OBJ
IF EMPTY? :OBJ [OP 0] [OP 1 + LENGTH BF→
:OBJ]
END
```

```
TO GREET
PR [GOOD MORNING. HOW ARE YOU TODAY?]
END
```

```
TO SPI :SIDE :ANGLE :INC
FD :SIDE
RT :ANGLE
SPI :SIDE + :INC :ANGLE :INC
END
```

```
ANIMAL IS AARDVARK
LENGTH IS 3.98
MYNAME IS SYLVIE
```

POLY, SPI, and LENGTH are in the *package* called SHAPES.

```
?POALL "SHAPES
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END
```

```
TO SPI :SIDE :ANGLE :INC
FD :SIDE
RT :ANGLE
SPI :SIDE + :INC :ANGLE :INC
END
```

```
LENGTH IS 3.98
```

POFILE

Command

Format: POFILE *filespec*

Remarks: The *filespec* is the name of a file you have previously stored on a disk, as explained under “Naming Files” in Chapter 4. POFILE (Print Out FILE) prints out the contents of a file. The output always goes to the screen.

An easy way to copy a file to another file is:

```
DRIBBLE "FILE2 POFILE "FILE1 NODRIBBLE
```

If you try to POFILE with a file that has already been opened with the OPEN command, the following error message is displayed:

```
FILE filespec ALREADY OPEN
```

In this case, close the file with CLOSE before you use POFILE.

POFILE Command

Examples: ?POFILE "CLASS
MARIE BARBEAU
SHARNEE CHAIT
JULIE DOLL

prints out the contents of the file called CLASS.

The DUMP procedure prints out the contents of a file on the printer as well as on the screen.

```
TO DUMP :FILENAME  
DRIBBLE "LPT1  
POFILE :FILENAME  
NODRIBBLE  
END
```

```
?DUMP "CLASS
```

PONS

Command

Format: PONS
PONS *package*
PONS *packagelist*

Remarks: Stands for “Print Out NameS”.

When PONS has no input, it prints the name and value of every variable in the workspace except the ones that are buried. See the BURY command. To put a variable into a package see the PPROP command.

POPS Command

Format: POPS
 POPS *package*
 POPS *packagelist*

Remarks: When POPS has no input, it prints the definition of every procedure in the workspace except the ones that are buried. See the BURY command.

 If an input *package* or *packagelist* is present, POPS prints only the definitions of procedures in that *package* or *packagelist* (including those that are in buried packages).

 Because POPS takes an optional input, you cannot put another command immediately after POPS on the same line.

POPS

Command

Examples: ?POPS

TO POLY :SIDE :ANGLE

FD :SIDE

RT :ANGLE

POLY :SIDE :ANGLE

END

TO SPI :SIDE :ANGLE :INC

FD :SIDE

RT :ANGLE

SPI :SIDE + :INC :ANGLE :INC

END

?POPS "LEFTOVER

TO LENGTH :OBJ

IF EMPTY? :OBJ [OP 0] [OP 1 + LENGTH BF→

:OBJ]

END

TO GREET

PR [GOOD MORNING. HOW ARE YOU TODAY?]

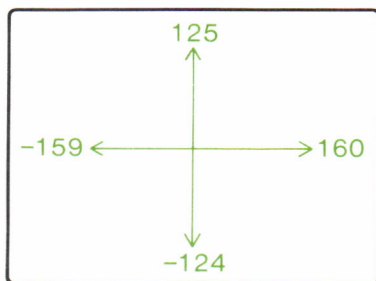
END

POS Operation

Format: POS

Remarks: Stands for “POSition”.

Outputs the coordinates of the current position of the turtle in the form of a list [x y]. When you start up Logo, the turtle is at [0 0], the center of the turtle field. The limits of the screen can be represented by the following diagram, if you have not changed the value of .SCRUNCH.



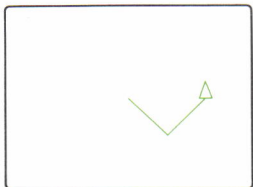
Examples: TO GOODVEE
MAKE "SAVEPOS POS
VEE
PENUP
SETPOS :SAVEPOS
PD
END

TO VEE
RT 135 FD 20
LT 90 FD 20
LT 45
END

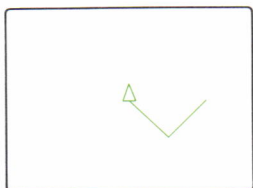
POS

Operation

GOODVEE remembers the turtle's starting position by storing it in the variable SAVEPOS. GOODVEE then calls the procedure VEE and restores the turtle's position to wherever it was at the start of GOODVEE.



?VEE



?CS

?GOODVEE

POTS Command

Format: POTS
POTS *package*
POTS *packagelist*

Remarks: Stands for “Prints Out TitleS”.

When POTS has no input, it prints the title line of every procedure in the workspace except the ones that are buried. See the BURY command.

If an input *package* or *packagelist* is present, POTS prints only the titles of procedures in that *package* or *packagelist* (including those that are buried).

Because POTS takes an optional input, you cannot put another command immediately after POTS on the same line.

Examples: ?POTS
TO POLY :SIDE :ANGLE
TO LENGTH :OBJ
TO GREET
TO SPI :SIDE :ANGLE :INC

?POTS "SHAPES
TO POLY :SIDE :ANGLE
TO SPI :SIDE :ANGLE :INC

?POTS [SHAPES LANG]
TO POLY :SIDE :ANGLE
TO SPI :SIDE :ANGLE :INC
TO LENGTH :OBJ

POWER

Operation

Format: `POWER n a`

Remarks: Outputs the number n raised to the power a . The result is rounded to PRECISION significant digits. When a is not an integer, PRECISION must be less than or equal to 100.

Note that:

- If n is 0, then a must be positive.
- If n is negative, then a must be an integer.
- When a is an integer, `POWER n a` is calculated using successive squarings and multiplications and is shorter to execute than when a is not an integer. In this latter case, `POWER n a` is the same as `EXP (a * LN n)` and requires the evaluation of a logarithm and an exponential.

Examples: `?SETPRECISION 5`
 `?PRINT POWER 10 50`
 `1.0000E+0050`

`?PRINT POWER 0 5`
 `0`

`?PRINT POWER 0 -5`
 `POWER DOESN'T LIKE [0 - 5] AS INPUT`

`?PRINT POWER 2 0.5`
 `1.4142`

`?PRINT SQRT 2`
 `1.4142`

POWER Operation

```
?SETPRECISION 40  
?PRINT POWER -2 50  
1125899906842624
```

```
?PRINT POWER 2 -50  
8.881784197001252323389053344726562500→  
000E-0016
```

```
?SETPRECISION 10  
?MAKE "P POWER 55 SQRT 3  
?PRINT :P  
1033.704966
```

```
?PRINT POWER :P SQRT 3  
166375.0011
```

```
?SETPRECISION 150  
?PRINT POWER 7 3  
343
```

```
?PRINT POWER 7 3.17  
POWER DOESN'T LIKE PRECISION 150
```

```
?SETPRECISION 25  
?PRINT POWER 3 -1  
0.33333333333333333333333333333333
```

```
?PRINT POWER 3 -2  
0.11111111111111111111111111111111
```

```
?PRINT POWER -2 0.5  
POWER DOESN'T LIKE [-2 0.5] AS INPUT
```

```
?PRINT POWER 3.21462 6.4179437  
1797.75081107492280180127
```

PPROP

Command

Format: PPROP *name prop object*

Remarks: Stands for “Put PROPerTy”.

Any Logo word (procedure, variable or package) may have a property list associated with it. A property list consists of an even number of elements. Each pair of elements consists of the name of the property and its value. The property list has the following form:

[PROP1 VAL1 PROP2 VAL2 ...]

The *name* input is the name of the procedure or variable.

The *prop* input is the name of the property.

The *object* input is the value of the property.

PPROP gives *name* the property *prop* with value *object*.

Once it is stored, the property value can be accessed by using GPROP and referring to the name of the property.

You can put a variable into a package by using PPROP to give a variable the VALPKG property.

(Note that ERALL, which erases procedure definitions and variables, does not reclaim the space used by properties of non-existent procedures that have been packaged by PPROP. Use REMPROP to erase properties.)

PPROP Command

Examples: ?SHOW PLIST "BIRD
[I.D. PHOENIX LEGS 2]
?PPROP "BIRD "SIZE [VERY BIG]
?SHOW PLIST "BIRD
[SIZE [VERY BIG] I.D. PHOENIX LEGS 2]

Put the variable VAR into the package NAMES by giving VAR the VALPKG property with the value NAMES.

```
?PPROP "VAR "VALPKG "NAMES  
?PPS "NAMES  
VAR'S VALPKG IS NAMES
```

PPS

Command

Format: PPS
PPS *package*
PPS *packagelist*

Remarks: Stands for “Print Properties”.

Prints the property lists as follows:

- With no inputs, prints the property lists of:
 - variables which are not in buried packages;
 - procedures which are not in buried packages;
 - all packages (buried or not) with a property list.
- With *package* or *packagelist* input, prints the property lists of everything in the named packages.

Examples: ?PPS

```
CHARACTERS'S BURY IS TRUE  
SPI'S PROCPKG IS SHAPES  
POLY'S PROCPKG IS SHAPES  
.SYSTEM'S BURY IS TRUE
```

CHARACTERS is the name of a buried package.
SPI and POLY are two procedures in the unburied package SHAPES.

To find out what is in the buried package
CHARACTERS, type PPS followed by the name of
the package.

```
?PPS "CHARACTERS  
CHARACTERS'S BURY IS TRUE  
BIRD'S I.D. IS PHOENIX  
BIRD'S COLOR IS YELLOW  
BIRD'S LEGS IS 2  
BIRD'S SIZE IS [VERY BIG]  
FROG'S I.D. IS FREDDY  
FROG'S COLOR IS GREEN  
FROG'S LEGS IS 4
```

PRECISION

Operation

Format: PRECISION

Remarks: Outputs the number of significant digits to which a number is rounded when used by Logo.

If you work at different precisions during one session, numbers having more than PRECISION digits may be in your workspace. The precision of these numbers won't show unless you change to a higher precision by using SETPRECISION.

When you start Logo, you are in precision 10. See Chapter 5 for further information.

Examples: ?PR PRECISION

10

?MAKE "A 12345678912345.1234

?PRINT :A

1.234567891E+0013

?MAKE "B 0.001234567891

?PRINT :B

1.234567891E-0003

?MAKE "D -0.000000123456789012312349

?PRINT :D

-1.234567890E-0007

PRECISION Operation

The output of an operation is always rounded to PRECISION significant digits.

```
?PRINT 1/3  
0.3333333333
```

```
?PRINT 3 * (1/3)  
0.9999999999
```

```
?PRINT 3/3  
1
```

```
?PRINT 2/3  
0.6666666667
```

```
?PRINT 2 * (1/3)  
0.6666666666
```

```
?PRINT 3 * 1/3  
1
```

PRIMITIVEP

Operation

Format: PRIMITIVEP *name*

Remarks: Outputs TRUE if *name* is the name of a primitive; otherwise FALSE.

Examples: ?PRINT PRIMITIVEP "FORWARD
TRUE
?PRINT PRIMITIVEP "SQUARE
FALSE

PRINT (PR) Command

Format: PRINT *object*
(PRINT *object1 object2 ...*)

Remarks: Prints its inputs, a word or a list, on the screen, followed by a carriage return. If you have opened a file or device for writing, then PRINT's input is sent to that file or device. See the SETWRITE command. The outermost brackets of lists are not printed. Compare PRINT with TYPE and SHOW.

If PRINT has more than one input, you must put parentheses around PRINT and its inputs.

Examples: ?PRINT "A
A
?PRINT "A PRINT [A B C]
A
A B C
?(PRINT "A [A B C])
A A B C
?PRINT []

?

Note that only one space is printed between words. If you want PRINT to print more spaces, you must use the \ (backslash).

```
?PRINT [HELLO  THERE]
HELLO THERE
?PRINT [HELLO \ \ \ THERE]
HELLO  THERE
```

PRINT (PR) Command

It should be noted that a backslash is not equivalent to a space. However, a backslash is needed to indicate that the next character is to be printed literally.

The following procedure prints a message a specified number of times.

```
TO REPRINT :MESSAGE :HOWMANY
IF :HOWMANY < 1 [STOP]
PR :MESSAGE
PR [ ]
REPRINT :MESSAGE :HOWMANY - 1
END
```

```
?REPRINT [TODAY IS FRIDAY!] 4
TODAY IS FRIDAY!
```

```
TODAY IS FRIDAY!
```

```
TODAY IS FRIDAY!
```

```
TODAY IS FRIDAY!
```

```
?
```

You can direct the output of PRINT to a file or device. In the following example, PRINT's output is sent to the printer. See Chapter 4, "File Handling" for more examples.

```
?OPEN "LPT1
?SETWRITE "LPT1
?PRINT [EVERYTHING GOES TO THE PRINTER]
?
```


PRODUCT Operation

Format: `PRODUCT $a\ b$`
 `(PRODUCT $a\ b\ c\ \dots$)`
 `PRODUCT a`

Remarks: Outputs the product of its inputs. Equivalent to $*$, an infix operation. If `PRODUCT` has more than two inputs, you must put parentheses around `PRODUCT` and its inputs.

Examples: `?PRINT PRODUCT 6 2`
 `12`

`?PRINT (PRODUCT 2 3 4)`
`24`

`?PRINT PRODUCT 2.5 4`
`10`

`?PRINT PRODUCT 2.5 2.5`
`6.25`

`?PRINT PRODUCT 6`
`6`

When b is not given, the second input is the multiplicative identity 1.

PRODUCT

Operation

Note that you need parentheses with 1 input if other commands follow on the Logo line.

The following procedure cubes its input.

```
TO CUBE :NUM  
  OP (PRODUCT :NUM :NUM :NUM)  
END
```

```
?PR CUBE 2  
8
```

QUOTIENT Operation

Format: QUOTIENT $a\ b$

Remarks: Outputs the result of dividing a by b

QUOTIENT 12 5

gives the same result as $12 / 5$. See the / operation.
See also the REMAINDER operation.

If b is 0, the following error message will be printed:

CAN'T DIVIDE BY ZERO

Examples: ?PRINT QUOTIENT 12 5
2.4

?PRINT QUOTIENT -12 5
-2.4

?PRINT QUOTIENT 6 2.5
2.4

?PRINT QUOTIENT 3.2 0
CAN'T DIVIDE BY ZERO

RANDOM Operation

Format: RANDOM n

Remarks: Outputs a random non-negative integer less than n .
The n can be any positive, real number. If n is not an integer, it is rounded.

Example: RANDOM 6 could output 0, 1, 2, 3, 4, or 5. The following program simulates the roll of a six-sided die (half a pair of dice).

```
TO DICE6  
OUTPUT 1 + RANDOM 6  
END
```

```
?PRINT DICE6  
3  
?PRINT DICE6  
5  
?PRINT DICE6  
3
```

READCHAR (RC) Operation

Format: READCHAR

Remarks: Outputs the first character read from the current read device or file. If no device or file is set, READCHAR reads from the keyboard. This character can even be a Ctrl character (except for Ctrl-Break).

If no character is waiting to be read, READCHAR waits until you type something. An exception occurs when the end-of-file position is reached on a file being read. In this case, READCHAR outputs a carriage return (an empty line).

Note that READCHAR does not echo what you type on the screen. See the KEYP operation.

Certain keys or key combinations, such as the cursor keys, that cannot be represented in standard ASCII code, return two characters.

The first character is the null character, a blank (ASCII code 00). This indicates that the next character is part of a special extended code and does not have its normal significance.

The second character allows you to determine the actual key that was pressed.

The extended codes are listed in Appendix D, "ASCII Character Code."

READCHAR (RC)

Operation

Example: The following procedure lets you run certain commands with a single keystroke (F does FORWARD 5, R does RIGHT 10, and L does LEFT 10—you can add to the list).

```
TO DRIVE
MAKE "CHAR READCHAR
IF :CHAR = "F [FD 5]
IF :CHAR = "R [RT 10]
IF :CHAR = "L [LT 10]
DRIVE
END
```

```
?DRIVE
```

Press F, R, or L to make a turtle drawing on the screen. You don't need to press Enter after each character.

READCHAR (RC) Operation

The GETEC procedure prints the ASCII code of the character you type as input. If you type certain keys, such as Cursor Up, two ASCII codes are printed.

```
TO GETEC  
  PRINT ASCII RC  
  PRINT ASCII RC  
END
```

```
?GETEC
```

Press the Cursor Up key, Logo prints

```
0  
72
```

The Cursor Up key generates an extended code consisting of two characters. Each RC reads one character. The ASCII operation is then used to convert each character to a number.

READCHARS (RCS)

Operation

Format: `READCHARS n`

Remarks: The *n* is the number of characters you want to read. `READCHARS` outputs the first *n* number of characters typed at the keyboard or read from the current read device or file.

If no characters are waiting to be read, `READCHARS` waits for you to type something.

An exception occurs when the end-of-file position is reached on a file being read. In this case, `READCHARS` outputs *n* carriage returns (empty lines).

Note that `READCHARS` does not echo what you type on the screen. Compare this with `READCHAR`.

Certain keys and key combinations output two characters, indicating a special extended code. For a discussion of extended codes, see the `READCHAR` operation. The extended codes are listed in Appendix D, “ASCII Character Codes.”

READCHARS (RCS) Operation

Example: `?PRINT READCHARS 4`

Press the following keys:

ABC (Don't press the Enter key.)
Nothing happens.

Now press

`D`

The following appears on the screen:

`ABCD`

READEOF

Operation

Format: READEOF

Remarks: Stands for "READ End Of File Position".

Outputs TRUE if the current file position is end-of-file; otherwise FALSE. You must open the file and SETREAD to the file before you can use READEOF. If you do not, Logo prints the error message:

NO FILE SELECTED

Examples: ?OPEN "TELNOS
?SETREAD "TELNOS
?PRINT READEOF
FALSE

If the file TELNOS has something in it, then READEOF is FALSE when the file TELNOS is just opened for reading.

If the file is empty then READEOF is TRUE.

The READFILE procedure reads a file that is already open until the end-of-file position is reached.

```
TO READFILE :FILE
  SETREAD :FILE
  IF READEOF [CLOSE :FILE STOP]
  PRINT READLIST
  READFILE :FILE
END
```

```
?OPEN "TELNOS
?READFILE "TELNOS
ALAN: 555 - 2654
FRANCOIS: 555 - 2222
```

READER Operation

Format: READER

Remarks: Outputs the current file specification or device which is open for reading. If the file is on the diskette in the default drive, then the drive letter is not included in the output. Compare this with the ALLOPEN operation. See "Data File" in Chapter 4 for information on data files.

Examples: ?PRINT READER
B:ADDRESS

The file called ADDRESS on the diskette in drive B is open for reading.

```
TO CHECKREAD :FILE  
IF NOT EQUALP READER :FILE [OPEN :FILE →  
SETREAD :FILE]  
IF READEOFP [CLOSE :FILE STOP]  
PRINT READLIST  
CHECKREAD :FILE  
END
```

```
?CHECKREAD "CLASS.LIS  
CAROLYN KIERAN  
JUDY LEGALLAIS  
JOHANNE GAMACHE
```

The CHECKREAD procedure checks if the file it has as input is currently open for reading. If not, it opens the file for reading. It then reads the file until the end-of-file position is reached. Then the file is closed, and execution of the procedure stops.

READLIST (RL)

Operation

Format: READLIST

Remarks: Reads a line of information that is sent from the current read device or file and outputs the information in the form of a list. Normally, this device is the keyboard, where a person types in the information followed by pressing the Enter key. This information is echoed on the screen. The command **SETREAD** allows you to set other read devices or files.

Examples: ?SHOW READLIST
TALK LOGO
[TALK LOGO]

The procedure GET.USER asks you for your name and then welcomes you to Logo.

```
TO GET.USER
PRINT [WHAT IS YOUR NAME?]
MAKE "USER READLIST
PRINT SE [WELCOME TO LOGO,] :USER
END
```

```
?GET.USER
WHAT IS YOUR NAME?
GUY
WELCOME TO LOGO, GUY
```

```
?GET.USER
WHAT IS YOUR NAME?
GUY TREMBLAY
WELCOME TO LOGO, GUY TREMBLAY
```

READPOS

Operation

Format: READPOS

Remarks: Stands for "READ POSition".

Outputs the current reader position in the file that is being read. To set the reader position, see the SETREADPOS command. If you have not opened a file to be read, Logo prints the error message:

NO FILE SELECTED

Examples: ?OPEN "TELNOS
?SETREAD "TELNOS
?PRINT READPOS
0

If you have just opened a file for reading, READPOS is 0.

The procedure LISTFILE lists the information stored in the read file, along with a number indicating where each line is stored.

```
TO LISTFILE :FILE
IF READEOFP [STOP]
PRINT READPOS
PRINT READLIST
LISTFILE :FILE
END
```

```
?OPEN "TELNOS
?SETREAD "TELNOS
?LISTFILE "TELNOS
0
ALAN: 555 - 2654
18
FRANCOIS: 631 - 2222
```

READWORD (RW)

Operation

Format: READWORD

Remarks: Reads a word sent from the current read device or file and outputs it. Normally, this device is the keyboard and READWORD waits for you to type and press the Enter key. If more than one word is typed, READWORD outputs only the first word. Therefore, what you type is echoed on the screen. If no word is typed before the Enter key is pressed, READWORD outputs an empty list.

If you use READWORD from a file, READWORD reads only the first word of the line, and then positions READPOS to the next line in the file.

When the end-of-file position is reached, READWORD outputs an empty list. See the READLIST, READCHAR, and READCHARS operations, and the SETREAD command.

READWORD (RW) Operation

Examples: ?SHOW READWORD
LONDON ONTARIO
LONDON

The following procedure asks your age and then prints how old you will be next year.

```
TO AGE  
PRINT [HOW OLD ARE YOU?]  
PRINT MESSAGE READWORD  
END
```

```
TO MESSAGE :AGE  
OP SE [NEXT YEAR YOU WILL BE] :AGE + 1  
END
```

```
?AGE  
HOW OLD ARE YOU?  
11  
NEXT YEAR YOU WILL BE 12
```

```
?AGE  
HOW OLD ARE YOU?  
35  
NEXT YEAR YOU WILL BE 36
```

RECYCLE

Command

Format: RECYCLE

Remarks: Performs a garbage collection, thereby freeing up as many nodes as possible. When you don't use RECYCLE, garbage collections happen automatically whenever necessary, but each one takes at least one second.

Running RECYCLE before a time-dependent activity prevents the automatic garbage collector from slowing a procedure down at an awkward time. Running RECYCLE before a very large program may prevent your running out of workspace. See the NODES command.

Misspellings, typing errors, and words that are no longer being used are not destroyed until you do a RECYCLE. All currently existing words can be seen on the list output by .CONTENTS. For instance, if you type

```
?PRIMT "FOO
I DON'T KNOW HOW TO PRIMT
?ALSJKDFLKDFJKLFJ
```

the words PRIMT, FOO, and ALSJKDFLKDFJKLFJ will be created. If you are running out of space and have a lot of these useless words, you should use RECYCLE. Note that the last word you have created will still exist in the .CONTENTS list even after doing RECYCLE.

REMAINDER Operation

Format: REMAINDER a b

Remarks: Divides a by b and outputs the remainder. The a and b can be any real numbers. If b is 0, Logo prints the error message:

CAN'T DIVIDE BY ZERO

Examples: ?PRINT REMAINDER 13 5
3

?PRINT REMAINDER 13 15
13

?PRINT REMAINDER -13 5
-3

?PRINT REMAINDER 13 5.5
2

?PRINT REMAINDER 13.5 2
1.5

?PRINT REMAINDER 13 0
CAN'T DIVIDE BY ZERO

REMAINDER

Operation

The following procedure tells you whether its input is even:

```
TO EVENP :NUMBER  
  OUTPUT 0 = REMAINDER :NUMBER 2  
END
```

```
?PRINT EVENP 5  
FALSE
```

```
?PRINT EVENP 12462  
TRUE
```

The following more general procedure tells whether its first input is a divisor of its second input.

```
TO DIVISORP :A :B  
  OUTPUT 0 = REMAINDER :B :A  
END
```

```
?PRINT DIVISORP 3 15  
TRUE
```

```
?PRINT DIVISORP 4 15  
FALSE
```

REMPROP Command

Format: `REMPROP name prop`

Remarks: Removes the property *prop* and its value from the property list of *name*.

Examples: `?SHOW PLIST "SHIP`
 `[SPEED 50 HEADING 40 COLOR GREEN]`

`?REMPROP "SHIP "HEADING`
`?SHOW PLIST "SHIP`
`[SPEED 50 COLOR GREEN]`

See also the PPROP command and the GPROP operation.

REPARSE

Command

Format: REPARSE

Remarks: When you define or erase a procedure, Logo may have to change the interpretation of many other procedures in your workspace. Like garbage collection (see the RECYCLE command), this reparsing happens automatically, so you don't have to worry about it. But, also like garbage collection, the automatic reparsing may slow down your program at an awkward time.

The REPARSE command can be used to make all needed reparsing happen right away. After you use it, you are safe from further reparsing at least until the next time you define or erase a procedure.

REPEAT Command

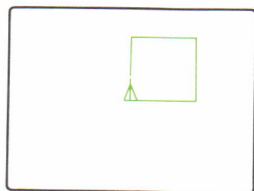
Format: REPEAT *n instructionlist*

Remarks: Runs a list of instructions the specified number of times. The *n* can be any positive, real number. See “Numbers That Logo Recognizes” in Chapter 5. If *n* is not an integer, it is rounded to an integer. If *n* is negative, the error message

REPEAT DOESN'T LIKE *n* AS INPUT

is printed.

Examples:



?REPEAT 4 [FD 50 RT 90]

draws a square 50 turtle steps on a side.

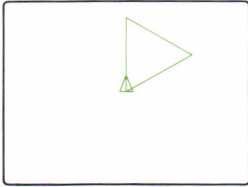
REPEAT 5 [PRINT RANDOM 20]

prints five random numbers from 0 through 19.

REPEAT Command

The following procedure draws polygons:

```
TO POLY :SIDE :ANG  
REPEAT 360/:ANG [FD :SIDE RT :ANG]  
END
```



```
?POLY 45 120
```

RERANDOM

Command

Format: RERANDOM

Remarks: Makes RANDOM act reproducibly. Once you run RERANDOM, Logo will remember the sequence of numbers obtained by using RANDOM. After that, each time you run RERANDOM, the result of using RANDOM will give the same sequence of random numbers.

Examples: The DICE procedure prints the rolls of a six-sided die for the number of throws you specify. The sequence of numbers is different each time.

```
TO DICE :THROWS
  IF :THROWS = 0 [STOP]
  PRINT 1 + RANDOM 6
  DICE :THROWS - 1
END
```

(Your numbers may differ from those printed here.)

```
?DICE 6
3
5
6
4
2
6
```

```
?DICE 6
4
3
2
6
3
5
```

RERANDOM

Command

Now run RERANDOM and DICE twice to reproduce the same sequence of dice throws.

```
?RERANDOM
```

```
?DICE 6
```

```
6
```

```
2
```

```
5
```

```
3
```

```
1
```

```
6
```

```
?RERANDOM
```

```
?DICE 6
```

```
6
```

```
2
```

```
5
```

```
3
```

```
1
```

```
6
```

Run DICE without RERANDOM to produce a new sequence of dice throws.

```
?DICE 6
```

```
5
```

```
1
```

```
2
```

```
4
```

```
2
```

```
3
```


RERANDOM Command

Now run RERANDOM and DICE again to reproduce the original sequence of dice throws.

```
?RERANDOM
```

```
?DICE 6
```

```
6
```

```
2
```

```
5
```

```
3
```

```
1
```

```
6
```

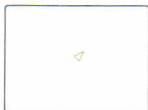
RIGHT (RT) Command

Format: RIGHT *degrees*

Remarks: Turns the turtle right (clockwise) the specified number of *degrees*. The *degrees* can be any real number from -9999 through 9999 .



CS

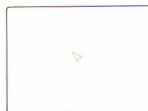


RIGHT 45

RIGHT 45 turns the turtle 45 degrees right.



CS



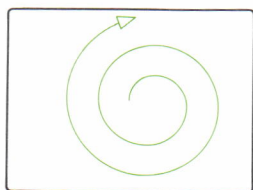
RIGHT -45

RIGHT -45 turns the turtle 45 degrees left.

RIGHT (RT) Command

Example: The following is a procedure that draws spirals.

```
TO SPI :SIDE :ANGLE :INC  
FD :SIDE RT :ANGLE  
SPI :SIDE + :INC :ANGLE :INC  
END
```



```
?SPI 5 10 .1
```

If you want to stop SPI, press Ctrl-Break.

ROUND

Operation

Format: `ROUND n`

Remarks: Outputs n rounded to the nearest integer. Compare this with the examples included with the INT operation.

Examples: `?PRINT ROUND 5.2129`
5

`?PRINT ROUND 5.5129`
6

`?PRINT ROUND .5`
1

`?PRINT ROUND -5.8`
-6

`?PRINT ROUND -12.3`
-12

`?PR ROUND (5.3 + 9.4)`
15

`?PR ROUND ((ROUND 5.3) + 9.4)`
14

ROUND Operation

The following procedure outputs the distance from the turtle's position to HOME, rounded to the nearest integer.

```
TO FROM.HOME  
OP ROUND SQRT SUM XCOR * XCOR YCOR * YCOR  
END
```

```
?FORWARD 33.5  
?PRINT FROM.HOME  
34
```

RUN

Command or Operation

Format: RUN *instructionlist*

Remarks: Runs the specified list of instructions as if it were typed in directly. If *instructionlist* is an operation, then RUN outputs whatever *instructionlist* outputs.

Examples: The following procedure simulates a calculator.

```
TO CALCULATOR
PRINT RUN READLIST
PRINT [ ]
CALCULATOR
END
```

```
?CALCULATOR
2 + 3
5
```

```
17.5 * 3
52.5
```

```
42 = 8 * 7
FALSE
```

```
REMAINDER 12 5
2
```

Press Ctrl-Break to stop.

RUN

Command or Operation

The WHILE procedure runs a list of instructions while a specified condition is true.

```
TO WHILE :CONDITION :INSTRUCTIONLIST
TEST RUN :CONDITION
IFFALSE [STOP]
RUN :INSTRUCTIONLIST
WHILE :CONDITION :INSTRUCTIONLIST
END
```

```
?RT 10
?WHILE [XCOR < 100] [FD 25 PR POS]
```

```
4.341 24.62
8.682 49.24
13.023 73.86
```

```
•
•
•
```

```
104.19 90.881
```

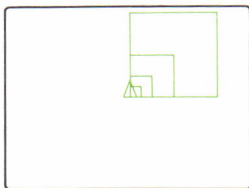
RUN

Command or Operation

The following procedure applies a command to each element of a list in turn.

```
TO MAP :CMD :LIST
IF EMPTY? :LIST [STOP]
RUN LIST :CMD WORD "" FIRST :LIST
MAP :CMD BF :LIST
END
```

```
TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END
```



```
?MAP "SQUARE [10 20 40 80]
```

```
?MAKE "NEW.ENGAND [ME NH VT MA RI CT]
```

```
?MAP "PRINT :NEW.ENGAND
```

ME

NH

VT

MA

RI

CT

RUN

Command or Operation

The procedure FOREVER repeats its input forever (unless it encounters an error or is stopped by pressing Ctrl-Break).

```
TO FOREVER :INSTRUCTIONLIST
RUN :INSTRUCTIONLIST
FOREVER :INSTRUCTIONLIST
END
```

The command

```
FOREVER [FD 1 RT 1]
```

tells the turtle to draw a circle.

The command

```
FOREVER [PR RUN READLIST PR [ ]]
```

is equivalent to the CALCULATOR procedure defined above.

RUN

Command or Operation

The procedure `SAFE.SQUARE` draws a square and then restores the pen mode (`PU`, `PD`, `PE`, or `PX`) to whatever it was previously.

```
TO SAFE.SQUARE  
MAKE "SAVETYPE FIRST PEN  
PENDOWN  
SQUARE 100  
RUN SE :SAVETYPE  
END
```

```
TO SQUARE :LEN  
REPEAT 4 [FD :LEN RT 90]  
END
```

```
?SHOW PEN  
[PENUP 3 1]  
?SAFE.SQUARE  
?SHOW PEN  
[PENUP 3 1]
```

The turtle draws a square with 100 turtle steps on each side.

```
RUN READLIST
```

runs any commands you type.

```
PRINT RUN READLIST
```

prints the output from any expression you type.

Format: SAVE *device*
 SAVE *filespec*
 SAVE *filespec package*
 SAVE *filespec packagelist*

Remarks: The *device* is any valid device and *filespec* is any valid file name as explained under “Naming Files” in Chapter 4. SAVE writes the workspace into the *device* or *filespec*. With *filespec* as input, SAVE creates a file called *file name*.LF on a disk in the default drive and saves in it procedures, variables, and properties.

If there is a second input to specify packages, SAVE saves the procedures, names, and properties included in the packages.

If there is no second input, the *entire* workspace is saved except for buried packages. See the BURY command.

If the file already exists, you must first erase it by using ERASEFILE; then SAVE can reuse that file name.

File names longer than eight characters are truncated to eight characters. Logo automatically adds .LF to program file names if an extension is not specified. You can override this extension by adding a period to the end of your file name or you can insert your own three-character extension.

See “Naming Files” in Chapter 4 for information on naming files and devices.

Note: You can save to any device that can be opened.

SAVE Command

Examples: `?SAVE "APRIL1`

saves your workspace in a file called APRIL1.LF

`?SAVE "APRIL1 "SHAPES`

saves the procedures, names, and properties from the package called SHAPES in the file APRIL1.LF

`?SAVE "APRIL1.`

saves your workspace in the file called APRIL1, with no extension.

`?SAVE "LPT1`

saves your workspace on to the printer, so everything is printed.

Format: SAVEPIC *device*
SAVEPIC *filespec*

Remarks: The *device* is the first parallel printer device, LPT1 or PRN, and *filespec* is a valid file name, as explained under “Naming Files” in Chapter 4.

Saves the screen (graphics and text) into the file named by *filespec*. If you have not specified an extension, SAVEPIC automatically adds the .PIC extension. If you don't want an extension, you must add a period to the end of the file name. The size of this file is 18398 bytes for the IBM Color/Graphics Monitor Adapter, and 4100 bytes for the IBM Monochrome Display.

The SAVEPIC command works in various ways:

- to a file saves the image of either a graphics or text screen.
- to an IBM Graphics Printer saves the image of either a graphics or text screen. If you have two screens (.SCREEN 2), SAVEPIC saves the graphics screen image.
- to any other printers saves the image of the text screen or the text portion of a graphics screen, if you have not loaded GRAPHICS when you started Logo.

Note: SAVEPIC to a printer has the same effect as pressing the Shift and PrtSc keys together.

SAVEPIC Command

You should hide the turtle by **HIDETURTLE** before doing **SAVEPIC**, otherwise when you load the picture you may have two turtles on the screen.

To load this file on the screen, see the **LOADPIC** command.

You cannot use **SAVEPIC** with the serial communications device names (**COM1** or **COM2**) in Logo.

```
?SAVEPIC "COM1  
SAVEPIC DOESN'T LIKE "COM1 AS INPUT
```

However, you can use the **DOS 2.00 MODE** command to redirect parallel printer output to the Asynchronous Communications Adapter. Then use **SAVEPIC "LPT1** to save text screens to the serial printer. See the **MODE** command in your *IBM Personal Computer DOS 2.00 Reference* manual.

Examples:

```
?HT  
?SAVEPIC "CAT
```

saves the screen in a file called **CAT.PIC**.

```
?TEXTSCREEN  
?SAVEPIC "LPT1
```

saves the text screen onto the printer.

You can also save your drawings which are on the graphics screen onto the printer, provided you have a graphics printer attached.

SENTENCE (SE) Operation

Format: SENTENCE *object1 object2*
(SENTENCE *object1 object2 object3 . . .*)
(SENTENCE *object1*)

Remarks: Outputs a list made up of the words in its inputs. When SENTENCE is used with a single input, parentheses around the expression are needed if there is something else on the line following the expression.

When SENTENCE is used with more than two inputs, you must enclose SENTENCE and its inputs in parentheses.

See the LIST operation for a chart comparing SENTENCE with other primitives that combine words and lists.

Examples: ?SHOW SENTENCE "PAPER "BOOKS
[PAPER BOOKS]

?SHOW SENTENCE [PAPER] [BOOKS]
[PAPER BOOKS]

?SHOW SENTENCE "APPLE [PEAR PLUM BANANA]
[APPLE PEAR PLUM BANANA]

?SHOW SENTENCE [TIME AND TIDE] [WAIT FOR
NO PERSON]
[TIME AND TIDE WAIT FOR NO PERSON]

?SHOW (SENTENCE "HOP "STEP "JUMP)
[HOP STEP JUMP]

?SHOW (SENTENCE "MONET)
[MONET]

?SHOW SENTENCE "MONET []
[MONET]

SENTENCE (SE)

Operation

The following procedure prints a compliment.

```
TO FLATTER :WHO
PR SE [HELLO THERE,] :WHO
PR [ ]
PR SE :WHO [IS A VERY NICE PERSON.]
PR SE WORD :WHO :WHO [IS DOUBLY NICE]
END
```

```
?FLATTER "MARIE
HELLO THERE, MARIE
```

```
MARIE IS A VERY NICE PERSON.
MARIEMARIE IS DOUBLY NICE
```

```
?MAKE "ANIMALS "KITTENS
?SHOW SENTENCE :ANIMALS
[KITTENS]
```

```
?SHOW (SENTENCE :ANIMALS) PR "PLAY
[KITTENS]
PLAY
```

Compare the outputs when SENTENCE and LIST are applied to lists that contain other lists.

```
?SHOW SENTENCE [THE DOG] [LIKES [RED MI→
CE]]
[THE DOG LIKES [RED MICE]]
```

```
?SHOW LIST [THE DOG] [LIKES [RED MICE]]
[[THE DOG] [LIKES [RED MICE]]]
```


SETBG Command

Format: SETBG *colornumber*

Remarks: Sets the background color in graphics mode to the color represented by *colornumber*, where *colornumber* is one of the following numbers:

0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	White	15	High-intensity White

Note that colors 8 through 15 are light or “high-intensity” values of colors 0 through 7.

Example: The following procedure cycles through all of the possible background colors.

```
TO CHANGEBG
IF BACKGROUND = 15 [SETBG 0]
WAIT 30
SETBG 1 + BG
CHANGEBG
END

?CHANGEBG
```

Press Ctrl-Break to stop.

SETCAPS

Command

Format: SETCAPS *pred*

Remarks: SETCAPS may be used to change Logo's handling of the Caps Lock key. Its input, *pred*, must be TRUE or FALSE. Because all Logo primitives are in capital letters, the Caps Lock key is disabled when Logo starts up. To enable the Caps Lock key and thus use the keyboard as a typewriter, the input to SETCAPS must be FALSE. See the CAPS operation.

Examples: ?SETCAPS "FALSE

enables the Caps Lock key. Press the Caps Lock key. The keyboard now can be used like a typewriter.

To return to the original setting, press the Caps Lock key again so all entries are uppercase. You can now reset CAPS to TRUE if you wish.

?SETCAPS "TRUE

disables the key again.

Remember: when the Caps Lock key is disabled, you get uppercase letters when you type. When you press the Shift key, you get lowercase letters.

SETCURSOR Command

Format: SETCURSOR *position*

Remarks: Sets the cursor to *position*. The first element of *position* is the line number; the second the column number.

Logo starts out with a screen WIDTH of 40, unless the IBM Monochrome Display is connected, in which case the WIDTH is 80. See the WIDTH operation.

Lines on the screen are numbered from 0 to 24 inclusive.

Columns are numbered from 0 to (WIDTH - 1) inclusive.

If WIDTH is 40 then the columns are numbered from 0 to 39. The last column is reserved for the continuation arrow, so you can set the cursor to column 38 but not to column 39.



If the line number of column number is out of range, or is not an integer, Logo prints the error message:

SETCURSOR DOESN'T LIKE *position* AS INPUT

SETCURSOR

Command

Examples: SETCURSOR [12 37] puts the cursor halfway down the right edge of the screen if you are using a monitor.

```
TO MOVECURSOR :X :Y
SETCURSOR LIST (:X + FIRST CURSOR) (:Y →
+ LAST CURSOR)
END
```

```
?PRINT "A MOVECURSOR 2 5 PRINT "B
```

MOVECURSOR moves the cursor the amount you specify for lines and columns.

```
TO LABELPIC :LABEL :POS
CT
LOCAL "CURSORPOS
MAKE "CURSORPOS CURSOR
SETCURSOR :POS
TYPE :LABEL
SETCURSOR :CURSORPOS
END
```

```
?CS
REPEAT 3 [FD 100 RT 120]
?LABELPIC "TRIANGLE [7 21]
```

LABELPIC prints text on the graphics portion of the screen.

Note: Text in graphics is erased with CS, not CT.

SETCURSOR

Command

Comments: The exact effect of placing text on the graphics screen when in the MIXEDSCREEN mode is complex. There is an interaction between text on the graphics screen and text on the textscreen. Thus, it is best to CLEARTEXT (CT) before placing text on the graphics screen.

```
?CT
```

```
?MS
```

```
?SETCURSOR [7 4] PRINT "HELLO SETCURSOR→  
[22 0]
```

To avoid confusion, it is best to reset the cursor to the text portion of the screen. Remember that the text will scroll up onto the textscreen behind the graphics portion of the screen. When it does, the letters on the graphics screen and whatever is directly under it on the textscreen will be mixed. (For example, try filling the textscreen with text and then placing text on the graphics screen.)

When you clear the textscreen each time you place text on the graphics screen, the effect is comparable to the command PENREVERSE. For example,

```
?CS
```

```
?CT
```

```
?SETCURSOR [7 4] PR "HELLO SETCURSOR [2→  
1 0]
```

```
?CT
```

```
?SETCURSOR [7 4] PR "HELLO SETCURSOR [2→  
1 0]
```

SETCURSOR

Command

If the text is always cleared under the graphics screen, printing the same text exactly on top of text on the graphics portion of the screen will erase the letters.

To replace text on the graphics screen, use LABELPIC.

```
?LABELPIC "PICTURE [7 4]
```

prints PICTURE.

```
?LABELPIC "PICTURE [7 4]
```

erases the word PICTURE.

```
?LABELPIC "NEW [7 4]
```

puts the word NEW on the screen.

See the STAMP command for another way to label pictures.

SETDISK Command

Format: SETDISK *drive*

Remarks: Tells Logo to set the drive to the *drive* unit A, B, C, D, etc.

Example: ?SETDISK "B:

sets the disk to drive B.

The current drive is now B. File handling commands will go to B, and you do not need to specify a drive.

?LOAD "SUPPLIES.ART

will load the file SUPPLIES.ART from the diskette in drive B.

SETHEADING (SETH)

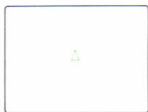
Command

Format: **SETHEADING** *degrees*

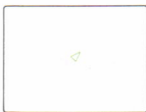
Remarks: Turns the turtle at its position so it is heading in the direction *degrees*. The *degrees* can be any real number from -9999 through 9999 . Positive numbers proceed clockwise from North.

Note that **LEFT** and **RIGHT** turn relative to the current heading of the turtle. **SETHEADING**, however, does not care what the current heading of the turtle is; **SETHEADING** sets the heading absolutely in terms of turtle degrees.

See the **HEADING** operation.

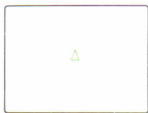


CS

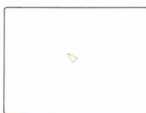


SETHEADING 45

SETHEADING 45 heads the turtle northeast.



CS



SETHEADING -45

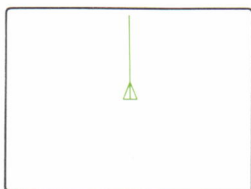
SETHEADING -45 heads the turtle northwest.

?PRINT HEADING
315

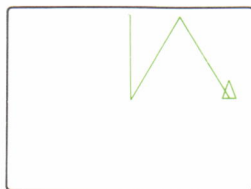
SETHEADING (SETH) Command

Example: The following procedure simulates a ball bouncing against a wall.

```
TO BOUNCE  
FORWARD 5  
IF YCOR > 90 [SETH 180 - HEADING]  
IF YCOR < 1 [SETH 0 STOP]  
BOUNCE  
END
```



?BOUNCE



?RT 30 BOUNCE

SETPAL

Command

Format: `SETPAL n`

Remarks: Selects the palette of colors that the turtle's pen will use. The n must be 0 or 1. The colors which are available when you choose each palette are as follows:

Color	Palette 0	Palette 1
1	Green	Cyan
2	Red	Magenta
3	Brown	White

When n is 0, the turtle's pen color can be green, red, or brown. When n is 1, the pen colors can be cyan, magenta, or white. When Logo starts, the palette is set to 1. See the PALETTE operation.

If you change the palette, all colors on the screen will change to match the new palette. The text on the graphics screen changes from white to brown when the palette is set to 0.

Examples: `?SETPAL 0`

The palette of colors is set to 0.

`?SETPC 3`
`?FD 30`

The turtle will draw with a brown pen color.

`?SETPAL 1`
`?FD 30`

The turtle will draw with a white pen color when the palette is set to 1.

SETPC Command

Format: SETPC *colornumber*

Remarks: Sets the color of the pen to *colornumber*, where *colornumber* is one of the following numbers. The actual color depends on which palette is set.

Color	Palette 0	Palette 1
1	Green	Cyan
2	Red	Magenta
3	Brown	White

When the turtle starts up, the palette is set at 1 and the pen color is set at 3. If the pen color does not look right on your screen, adjust your monitor's tint control. However, when two lines of different colors are horizontally close to each other, one of them may be the wrong color no matter what you do. See the SETPAL command to change the palette of colors.

Note that *colornumber* 0 will set the pen color to the same shade as the background color.

Examples: ?SETPC 2
 ?FD 30

The turtle draws with a magenta pen color.

?SETPAL 0
?FD 30

The turtle draws with a red pen color.

SETPEN

Command

Format: SETPEN *penlist*

Remarks: Sets the pen state to the three-word *penlist*. The form of the input is the same as that of the output from PEN. See the PEN operation. The first word must be PENDOWN, PENERASE, PENUP, or PENREVERSE. The second word is the number representing the current pen color. The third word is the current palette number.

Examples: ?SETPEN [PENDOWN 3 1]

is equivalent to

?PENDOWN

?SETPC 3

?SETPAL 1

In the following example, the RESTORE procedure saves the current pen state and calls the DESIGN procedure.

The DESIGN procedure draws three sides of a square in three different colors. It returns to the RESTORE procedure with pen color cyan and the pen in the PENUP state.

SETPEN Command

The RESTORE procedure then restores the original pen state.

```
TO RESTORE :PROCEDURE  
MAKE "SAVEPEN PEN  
RUN SE :PROCEDURE  
SETPEN :SAVEPEN  
END
```

```
TO DESIGN  
PIECE 3  
PIECE 2  
PIECE 1  
PENUP FD 50  
END
```

```
TO PIECE :COLOR  
SETPC :COLOR  
FD 50 RT 90  
END
```

```
?RESTORE "DESIGN
```

SETPOS

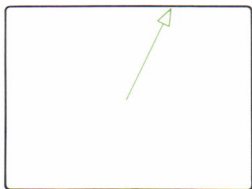
Command

Format: SETPOS *position*

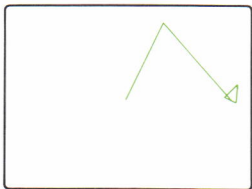
Remarks: Moves the turtle to the *position* indicated by a list of two numbers [x y]. See the POS operation. If the pen is down, the turtle draws a line as it moves to the new position.

The first element of the list is the x-coordinate, and the second, the y-coordinate. Inputs are rounded to precision 5 with a maximum of 3 digits after the decimal point. The turtle's heading is not changed by this command.

Examples: SETPOS [160 0] moves the turtle to a point halfway down the right edge of the screen.



?RT 30
?FD 120



?SETPOS [160 0]

SETPOS Command

The following procedure draws a box at a random position on the screen.

```
TO RANDOMBOX
MAKE "X NUMBER
MAKE "Y NUMBER
PU
SETPOS SENTENCE :X :Y
PD
REPEAT 4 [FD 25 RT 90]
END

TO NUMBER
MAKE "NUM RANDOM 100
IF 1 = RANDOM 2 [OUTPUT -:NUM]
OUTPUT :NUM
END

?RANDOMBOX
```

SETPRECISION

Command

Format: SETPRECISION *n*

Remarks: Sets the current precision to *n*. You can set the precision to any integer not less than 5 or greater than 1000. Note that as *n* increases, so does the time Logo takes to complete the computation.

Some primitives (SIN, COS, ARCTAN, PI, LN, and EXP) do not allow PRECISION greater than 100. Numbers used for graphics primitives are rounded to PRECISION 5 with a maximum of 3 digits after the decimal point. SETPRECISION does not affect the contents of your workspace.

The default precision when Logo starts is 10.

While doing arithmetic, choose the precision you want and avoid changing it to a lower precision setting. You may get surprising results due to roundings performed at various precisions. See “PRECISION and Logo Numbers” in Chapter 5 for more information.

Examples: TO ESTIMATE :N :M
 MAKE "REALPRECISION PRECISION
 SETPRECISION :M
 TYPE :N PRINT "?
 SETPRECISION :REALPRECISION
 END

 ?ESTIMATE 12*12 1
 SETPRECISION DOESN'T LIKE 1 AS INPUT IN→
 ESTIMATE:

 ?ESTIMATE 12345 * 12346 5
 1.5241E0008?

SETREAD Command

Format: SETREAD *device*
 SETREAD *filespec*

Remarks: The *device* is any valid device and *filespec* is any valid file name as explained under “Naming Files” in Chapter 4. SETREAD sets the *device* or *filespec* from which to receive input. After this command is given, READLIST, READWORD, READCHAR, and READCHARS read the information from this *device* or *filespec*.

Before you use SETREAD, you must open the *device* or *filespec* by the OPEN command. If you do not, Logo prints the error message:

FILE *filespec* IS NOT OPEN

If you try SETREAD to the printer or serial communication line, Logo prints the error message:

CAN'T READ DEVICE

SETREAD sets the reader position at the beginning of a file if the input is a file name. To write information to a file, use the SETWRITE command. For more information on reading from and writing to files, see Chapter 4, “File Handling.”

SETREAD

Command

Examples: ?OPEN "TELNOS
?SETREAD "TELNOS
?PRINT READPOS
0

The reader position is now at the beginning of the file.

?PRINT READLIST
LINDA: 555 - 6299

To set the reader back to the keyboard, the default mode, use the device name CON as the input to SETREAD.

?SETREAD "CON
?PRINT READLIST

Now READLIST waits for the list to be typed at the keyboard.

?SETREAD []
?PRINT READLIST

SETREAD with the empty list has the same effect as SETREAD "CON.

SETREADPOS

Command

Format: SETREADPOS *filepos*

Remarks: The *filepos* is the position of the file in bytes, where each byte is equivalent to one character. It can be a number from 0 to the total length of the file. SETREADPOS sets the file position of the file that is currently being read by SETREAD. When reading a file, you can check your file position by using the READPOS operation. The start of the file is position 0.

If you try to position the file beyond the end-of-file position, Logo prints the error message:

POSITION OUT OF RANGE

See the READPOS operation for more information.

SETREADPOS

Command

Examples: ?OPEN "TELNOS
?SETREAD "TELNOS
?SETREADPOS 2
?PRINT READCHAR
Y

The file TELNOS is opened to read. The reader is positioned at 2, and the character at that position is printed.

```
TO FILERL :POS
SETREADPOS :POS
OUTPUT READLIST
END
```

```
?PRINT FILERL 34
JUDY
```

The FILERL procedure outputs the list found at the file position you give as input.

SETSHAPE Command

Format: SETSHAPE *n*

Remarks: Changes the image carried by the turtle to *n*, one of 256 nonrotating patterns found in the ASCII table in Appendix D. Patterns 0 through 127 cannot be altered, while those from 128 through 255 may be replaced by using the SNAP command. To make the turtle “normal” again, use SETSHAPE “TURTLE”. See the SHAPE operation to determine the turtle’s current shape.

Note that TURTLE is a special word that stands for the triangular shape of the turtle. It has meaning only for SETSHAPE.

Example: TO DUMBTURTLE
 SETSHAPE 1 WAIT 45 FORWARD 8 STAMP
 SETSHAPE 2 WAIT 45 FORWARD 8 STAMP
 DUMBTURTLE
 END

 ?DUMBTURTLE

Press Ctrl-Break to end the procedure.

SETTC

Command

Format: SETTC *colorlist*

Remarks: Stands for “SET Text Color.”

SETTC takes a list of two numbers as its input.

The first number sets the foreground, the color in which characters are displayed on the screen.

The second number sets the background color of each character.

- If text is displayed in TEXTSCREEN mode, then the foreground colors can be 0 through 15.

Background colors can be 0 through 7, while 8 through 15 may be used to make characters blink as the cursor usually does.

The following is a table of numbers and their corresponding colors:

0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Brown	14	Yellow
7	White	15	High-intensity White

For monochrome screen:

Foreground:	0	black when used with background 7 or 15
	1	underlined white
	2-7	white
	8-15	high-intensity white (except with background 0, 7, 8, or 15)
Background:	0-6	black
	7	white when used with foreground 0 or 8
	8-15	blinking

- If text is displayed on the graphics screen (FULLSCREEN or MIXEDSCREEN mode), the foreground colors are limited to the turtle's pen colors. If the foreground color is greater than 3, the text color is set to the remainder of the foreground number divided by 4.

Note that color 0 is the background color.

Color	Palette 0	Palette 1
1	Green	Cyan
2	Red	Magenta
3	Brown	White

The second number (background) is ignored.

SETTC

Command

Examples: `?TEXTSCREEN`
`?SETTC [15 0] PRINT "GOTCHA!"`

will give white letters on a black background.

`?SETTC [4 8] PRINT "DANGER!"`

will give blinking red letters on a black background.
On a monochrome screen, the characters almost will be invisible.

SETTEXT Command

Format: SETTEXT *n*

Remarks: Sets the text portion of the graphics screen to *n*, the number of lines of text that will appear. The *n* can be any integer from 1 through 25. The number of lines starts from the bottom of the screen.

When Logo starts up, the graphics screen allows six lines of text at the bottom of the screen. The command MIXEDSCREEN uses the current amount of lines from SETTEXT to determine the amount of text you can use on the graphics screen.

The SETTEXT command switches you to the graphics screen if you are currently displaying the textscreen.

If .SCREEN is 2, then SETTEXT has no effect. See the .SETSCREEN command for details.

Example: ?SETTEXT 10

sets the text portion of the screen to the bottom 10 lines.

SETWIDTH

Command

Format: SETWIDTH *n*
 SETWIDTH *n a*

Remarks: Sets the width of the screen to *n* characters per line. The *n* can be any number from 2 through 80. The default setting for width is 40 on a monitor or TV and 80 on an IBM Monochrome Display.

The second input, *a*, specifies the amount you want to shift the display from its current position.

The *a* can be any positive or negative integer from 1 through 10. Input *a* has no effect if you are using an IBM Monochrome Display.

Input *a* is useful if you cannot see part of the beginning or ending of a line. If you want to shift your text *a* columns to the right, *a* is a positive number. If you want to shift your text *a* columns to the left, *a* is a negative number.

You can't do graphics with a line width of over 40 characters. In addition, the F2 and F4 function keys have no effect at top level. In order to do graphics again, you must set the width to 40 or less. If you are working with two screens (.SCREEN is 2), there is no restriction on the screen width for doing graphics.

SETWIDTH

Command

Examples: `?SETWIDTH 36`

sets the line width to 36 characters.

`?SETWIDTH 38 2`

sets the line width to 38 characters and shifts the text two characters to the right.

`?SETWIDTH 40 -2`

sets the line width to 40 characters and shifts the text two characters to the left from its current position, thus restoring the original condition.

Warning: The second input to SETWIDTH is a hardware correction. Using SETWIDTH carelessly may destroy your workspace and require you to restart Logo.

SETWRITE

Command

Format: SETWRITE *device*
 SETWRITE *filespec*

Remarks: The *device* is any valid device and *filespec* is any valid file name as explained under “Naming Files” in Chapter 4. SETWRITE sets the destination of inputs to PRINT, TYPE, SHOW, PO, and POFILE to *device* or *filespec*. You cannot use SETWRITE with a file or device that was not opened previously by using OPEN. If you try to do so, Logo prints the error message:

FILE *filespec* IS NOT OPEN

If you SETWRITE to a file, the file position is at the end of the file.

SETWRITE will only write to one file at a time. If SETWRITE is executed more than once, the last file given is the file that is written to.

Examples: ?OPEN "LPT1
 ?SETWRITE "LPT1

Now the printer is open for sending data.

?PRINT [LOGO TELEPHONE DIRECTORY]

This line is printed on the printer. In order to direct PRINT to the screen (and not to the printer), use SETWRITE with the empty list as input or SETWRITE “CON, where CON stands for console.

SETWRITE Command

The STORE procedure opens a file for writing and prints data into the file.

```
TO STORE :FILE :DATA  
OPEN :FILE  
SETWRITE :FILE  
PRINT :DATA  
CLOSE :FILE  
END
```

```
?STORE "TELNOS [ROBERT 555-2513]
```

You also may write to a serial device, such as a printer. The inputs to .SETCOM may be different for your serial device.

```
.SETCOM 1 300 2 7 1
```

This sets up the first serial communications line, at 300 baud, with even parity, 7 databits and 1 stop bit. See the .SETCOM command for details.

```
OPEN "COM1  
SETWRITE "COM1
```

Now the output of any PRINT commands will be directed to the serial printer.

SETWRITEPOS

Command

Format: SETWRITEPOS *filepos*

Remarks: The *filepos* is the file position from 0 to the end-of-file position (given by FILELEN) of the file being written to. SETWRITEPOS sets the file position for writing into the current file. This command is most useful when modifying a data file. If you try to use SETWRITEPOS with a device like the printer, Logo prints the error message:

CAN'T POSITION DEVICE

To check the current position, use the operation WRITEPOS. If you try to SETWRITEPOS to a position greater than the maximum length of the file, Logo prints the error message:

POSITION OUT OF RANGE

For more information and examples on data files, see “Data File” in Chapter 4.

Example: ?OPEN "BOOKLIST
?SETWRITE "BOOKLIST
?SETWRITEPOS 0
?PRINT [MINDSTORMS, Seymour Papert]

The file BOOKLIST is open for writing. SETWRITEPOS sets the position to the beginning of the file.

Format: SETX x

Remarks: Moves the turtle horizontally to a point with x-coordinate x (y-coordinate is unchanged). If the turtle's pen is set to draw, it will leave a trace.



CS

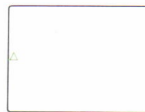


SETX -150

SETX -150 moves the turtle horizontally over to the left edge of the screen.



CS



PU SETX -150

SETX

Command

Example: The following procedure places the turtle at a random position on the screen.

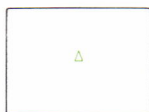
```
TO PLACET
  PENUP
  SETX NUMBER
  SETY NUMBER
  PENDOWN
END

TO NUMBER
  MAKE "NUM RANDOM 100
  IF 1 = RANDOM 2 [OUTPUT -:NUM]
  OUTPUT :NUM
END

?PLACET
```


Format: SETY y

Remarks: Moves the turtle vertically to a point with y -coordinate y (x -coordinate is unchanged).



CS

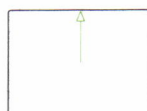


SETY -120

SETY -120 moves the turtle vertically down to the lower edge of the screen.



CS SETY 30



SETY 2 * YCOR

Example: The BOUNDS procedure draws a fence within the edges of the screen.

```
TO BOUNDS
PENUP SETX -150
SETY -120 PENDOWN
REPEAT 2 [FD 240 RT 90 FD 300 RT 90]
END
```

?BOUNDS

SHAPE

Operation

Format: SHAPE

Remarks: Outputs the number representing the shape that the turtle is currently set to by SETSHAPE. If the turtle is in its normal triangular shape, SHAPE outputs TURTLE. See the SETSHAPE command for information on changing the turtle's shape.

Examples: ?PRINT SHAPE
 TURTLE

When Logo starts up, the turtle is in its normal shape.

```
?SETSHAPE 65
?PRINT SHAPE
65
```

You changed the turtle's shape to an A (shape number 65).

SHOW Command

Format: **SHOW** *object*

Remarks: Prints *object* on the screen, and the cursor goes to the beginning of the next line. If *object* is a list, it is printed with brackets around it. Compare with the **TYPE** and **PRINT** commands.

Examples: ?SHOW "A
 A

 ?SHOW "A SHOW [A B C]
 A
 [A B C]

 ?TYPE "A TYPE [A B C]
 AA B C?

 ?PRINT "A PRINT [A B C]
 A
 A B C

 ?SHOW PLIST ".SYSTEM
 [BURY TRUE]

SHOWNP

Operation

Format: SHOWNP

Remarks: Outputs TRUE if the turtle is visible; otherwise FALSE.

Examples: ?HIDETURTLE
 ?PRINT SHOWNP
 FALSE

Here is a procedure that hides the turtle before drawing a circle, and then shows the turtle if it was visible before the procedure was run.

```
TO FASTCIRCLE :STEP
LOCAL "SHOWN
MAKE "SHOWN SHOWNP
HIDETURTLE
REPEAT 360 [FD :STEP RT 1]
IF :SHOWN [SHOWTURTLE]
END
```

```
?FASTCIRCLE 1
```

SHOWTURTLE (ST) Command

Format: SHOWTURTLE

Remarks: Makes the turtle visible; the opposite of
 HIDETURTLE.

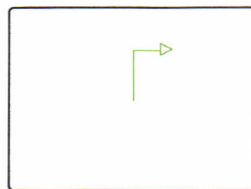
Examples:



?HIDETURTLE
?FD 50



?SHOWTURTLE



?RT 90
?FD 20

SIN

Operation

Format: SIN a

Remarks: Outputs the sine of an angle of a degrees. The result is rounded to PRECISION significant digits, where PRECISION must be less than or equal to 100. The number a may be any real number.

If PRECISION is greater than 100, Logo prints the error message:

SIN DOESN'T LIKE PRECISION n

Examples: ?SETPRECISION 10

 ?MAKE "A 36045

 ?PRINT SIN :A

 0.7071067812

 ?PRINT SIN 45

 0.7071067812

 ?PRINT :A

 36045

 ?PRINT SIN 0

 0

 ?PRINT SIN 90

 1

 ?PRINT SIN 89.995

 .9999999962

 ?PRINT SIN 30

 0.5

 ?PRINT SIN -30

 -0.5

SIN Operation

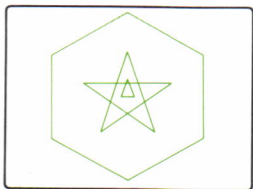
```
?SETPRECISION 50
?MAKE "S SIN 45
?PRINT :S
0.7071067811865475244008443621048490392→
8483593768847
```

```
?SETPRECISION 150
?PRINT SIN 63
SIN DOESN'T LIKE PRECISION 150
```

The following procedure draws polygons centered around the midpoint of the screen.

```
TO POLYCEN :SIDE :ANGLE
LOCAL "TURN LOCAL "RAD
MAKE "TURN :ANGLE + ((180-:ANGLE)/2)
MAKE "RAD (:SIDE/2) / (SIN :ANGLE/2)
PU FD :RAD RT :TURN PD
POLY :SIDE :ANGLE HEADING
LT :TURN PU BK :RAD PD
END
```

```
TO POLY :SIDE :ANGLE :START
FD :SIDE RT :ANGLE
IF :START = HEADING [STOP]
POLY :SIDE :ANGLE :START
END
```



```
?POLYCEN 50 60
?POLYCEN 50 144
```

SNAP

Command

Format: SNAP *n*

Remarks: Replaces the current turtle shape with whatever pattern happens to be underneath the turtle on the screen.

The *n* must be a shape number between 128 and 255 as defined in the ASCII table, Appendix D.

Whatever pattern is snapped will be stored in that “shape box” and will be accessible in the future by using **SETSHAPE**. Only the original turtle shape can show rotation. **SETSHAPE** “TURTLE” will give the turtle its original shape back.

Examples: This procedure defines a “diamond back” turtle.

```
TO LIST.OF.DOTS :DL
IF EMPTY? :DL [STOP]
DOT SE FIRST :DL FIRST BUTFIRST :DL
LIST.OF.DOTS BUTFIRST BUTFIRST :DL
END
```

```
TO DIAMONDBACK
CS HT
LIST.OF.DOTS [-3 0 -2 1 -1 2 0 3 1 2]
LIST.OF.DOTS [2 1 3 0 2 -1 1 -2 0 -3]
LIST.OF.DOTS [-1 -2 -2 -1 -3 -1 -3 -2]
LIST.OF.DOTS [-3 -3 3 -1 3 -2 3 -3]
SNAP 255
END
```

```
?DIAMONDBACK
```

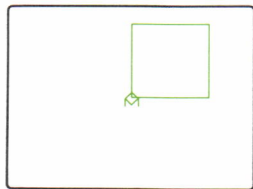
will give you a non-rotating turtle with this shape.

SNAP Command



The shape will be available by using

SETSHAPE 255



?CS

?ST

?SETSHAPE 255

?REPEAT 4 [FD 50 RT 90]

SQRT

Operation

Format: SQRT *n*

Remarks: Outputs the square root of *n*. If *n* is negative, Logo prints

SQRT DOESN'T LIKE *n* AS INPUT

Examples: ?PRINT SQRT 25
5

?PRINT SQRT 259
16.09347694

The following procedure outputs the distance from the turtle's position to HOME.

```
TO FROM.HOME
OP SQRT SUM XCOR * XCOR YCOR * YCOR
END
```

The procedure DISTANCE takes any two positions as inputs. It then outputs the distance between them.

```
TO DISTANCE :POS1 :POS2
OUTPUT SQRT SUM SQ (FIRST :POS1) - (FIRST :POS2)
ST :POS2) SQ (LAST :POS1) - (LAST :POS2)
)
END
```

```
TO SQ :N
OUTPUT :N * :N
END
```

```
?PRINT DISTANCE [-70 10] [50 60]
130
```

STAMP Command

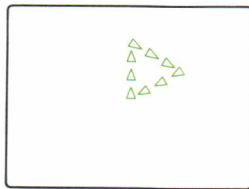
Format: STAMP

Remarks: Leaves the image of the current turtle shape on the screen in the current pen color. If the turtle is over its stamp, the turtle and the stamp become invisible until the turtle moves. Hiding the turtle with **HIDETURTLE** allows you to see the stamp. See the **SNAP** and **SETSHAPE** commands.

Examples: The turtle's shape is stamped before the turtle moves forward.

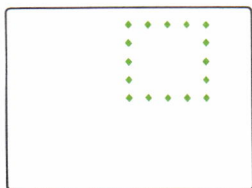
```
TO TRAIL :DIST :INC  
REPEAT :DIST/:INC [STAMP PU FD :INC]  
FD REMAINDER :DIST :INC  
END
```

The **TRAIL** procedure stamps the turtle in its pen color for the distance specified by the first input at intervals specified by the second input.



```
?REPEAT 3 [TRAIL 60 20 RT 120]  
?HT
```

STAMP Command



```
?ST  
?CS  
?SETSHAPE 4  
?REPEAT 4 [TRAIL 80 20 RT 90]  
?HIDETURTLE
```

The following procedure allows you to label pictures on the graphics screen.

```
TO LABELART :WORD  
IF EMPTY? :WORD [SETSHAPE "TURTLE HT ST→  
OP]  
SETSHAPE ASCII FIRST :WORD  
STAMP PU FD 8 PD  
LABELART BF :WORD  
END
```

```
?CS ST  
?RT 90  
?LABELART "PICTURE
```

STOP Command

Format: STOP

Remarks: Stops the procedure that is running and returns control to the caller. This command is meaningful only when it is within a procedure, not at top level. Note that a procedure containing **STOP** is a command. Compare with the **OUTPUT** operation.

Example:

```
TO FOREVER :MESSAGE :NUM
  IF :NUM = 0 [STOP]
  PR :MESSAGE
  FOREVER :MESSAGE :NUM - 1
END

?FOREVER [IBM LOGO] 4
IBM LOGO
IBM LOGO
IBM LOGO
IBM LOGO
```

SUM

Operation

Format: SUM *a*
SUM *a b*
(SUM *a b c ...*)

Remarks: Outputs the sum of its inputs. Equivalent to +, an infix operation. If SUM has more than two inputs, SUM and its inputs must be enclosed in parentheses.

Examples: ?PRINT SUM 5 2
7

?PRINT (SUM 1 3 2 -1)
5

?PRINT SUM 2.3 2.561
4.861

?PRINT SUM 12.5
12.5

The following procedure gives an arithmetic example:

```
TO ADD
MAKE "A RANDOM 20
MAKE "B RANDOM 20
(PRINT [HOW MUCH IS] :A "+" :B)
MAKE "ANSWER READWORD
IF :ANSWER = SUM :A :B [PRINT [THAT'S R→
IGHT]] [PRINT [THAT'S WRONG]]
END

?ADD
HOW MUCH IS 5 + 12
17
THAT'S RIGHT
```

TEST Command

Format: TEST *pred*

Remarks: Determines whether *pred* (a predicate) is TRUE or FALSE and remembers it for subsequent use by IFTRUE, IFFALSE, or both. Each TEST is local to the procedure in which it occurs.

Examples: The following procedure checks whether your response is FINE to the question HOW ARE YOU?

```
TO SHORTQUIZ  
PR [HOW ARE YOU?]  
TEST READLIST = [FINE]  
IFTRUE [PRINT [I'M GLAD TO HEAR IT]]  
END
```

```
?SHORTQUIZ  
HOW ARE YOU?  
TIRED
```

```
?SHORTQUIZ  
HOW ARE YOU?  
FINE  
I'M GLAD TO HEAR IT
```

TEXT

Operation

Format: TEXT *name*

Remarks: Outputs the definition of *name* (the name of a procedure) as a list of lists, which is suitable for input to DEFINE.

Examples: ?SHOW TEXT "POLY
[[SIDE ANGLE] [FD :SIDE RT :ANGLE] [POL→
Y :SIDE :ANGLE]]

The first element of the output is a list of the names of the procedure's inputs (without their colons). The rest of the elements are lists; each one is a line in the procedure definition. (If the procedure name is undefined, TEXT outputs the empty list.) The example above corresponds to

```
?PO "POLY
TO POLY :SIDE :ANGLE
FD :SIDE RT :ANGLE
POLY :SIDE :ANGLE
END
```

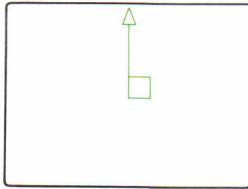
TEXT can be used with DEFINE to create procedures that modify other procedures. For example,

```
?PO "SQUARE
TO SQUARE
REPEAT 4 [FD 30 RT 90]
END
```


TEXT Operation

```
?DEFINE "SQUARE.WITH.TAIL LPUT [FD 100]→  
TEXT "SQUARE
```

```
?PO "SQUARE.WITH.TAIL  
TO SQUIRE.WITH.TAIL  
REPEAT 4 [FD 30 RT 90]  
FD 100  
END
```



```
?SQUARE.WITH.TAIL
```

An empty list is used to show that there are no inputs to a procedure.

```
?SHOW TEXT "SQUARE  
[[ ] [REPEAT 4 [FD 30 RT 90]]]
```

TEXTCOLOR (TC)

Operation

Format: TEXTCOLOR

Remarks: Outputs a list of the current text colors. The first number is the color of the characters (foreground) and the second number is the background color. See the SETTC command to change text colors. The following is a table of numbers and their corresponding colors:

Note: For Color Display background text only
0-7 are color choices; inputs of 8-15 give
blinking text colors.

0 Black	8 Gray
1 Blue	9 Light Blue
2 Green	10 Light Green
3 Cyan	11 Light Cyan
4 Red	12 Light Red
5 Magenta	13 Light Magenta
6 Brown	14 Yellow
7 White	15 High-intensity White

TEXTCOLOR (TC) Operation

For monochrome screen:

Foreground:	0	Black
	1	Underlined White
	2- 7	White
	8-15	High-intensity White
Background:	0- 6	Black
	7	White
	8-15	Blinking

Example: ?SHOW TEXTCOLOR
[7 0]

The text colors are white characters on a black background when Logo starts up.

TEXTSCREEN (TS)

Command

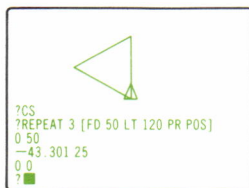
Format: TEXTSCREEN

Remarks: Devotes the entire screen to text. The turtle field will be temporarily invisible to you until a graphics procedure is run. The F1 function key is equivalent to TEXTSCREEN with one difference: The F1 function key can be used while a procedure is still running. To type TEXTSCREEN, you have to wait until you get the prompt. If you press the F1 key while a graphics procedure is running, the screen will switch momentarily to text and then switch right back to graphics. See the MIXEDSCREEN and FULLSCREEN commands.

If .SCREEN is 2, neither TEXTSCREEN nor the F1 function key will have any effect.

TEXTSCREEN (TS) Command

Examples:



?MIXEDSCREEN

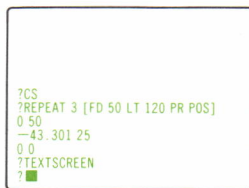
?CS

?REPEAT 3 [FD 50 LT 120 PR POS]

0 50

-43.301 25

0 0



?TEXTSCREEN

THING

Operation

Format: THING *name*

Remarks: Outputs the thing (or value) associated with the variable *name*. THING “ANY is equivalent to :ANY. The variable can be created by using the command MAKE or by defining a procedure with inputs. An important use of THING is to get the value of an operation that outputs a variable.

Examples: ?MAKE "WINNER "COMPUTER
 ?MAKE "COMPUTER [100 POINTS]
 ?PRINT THING "WINNER
 COMPUTER

 ?PRINT :WINNER
 COMPUTER

 ?PRINT THING :WINNER
 100 POINTS

The following procedure prints the contents of each variable in a list.

```
TO PRINTVARS :L
IF EMPTY? :L [STOP]
PR THING FIRST :L
PRINTVARS BF :L
END

?PRINTVARS [WINNER COMPUTER]
COMPUTER
100 POINTS
```

THING Operation

This procedure increments (adds 1 to) the value of a variable.

```
TO INC :X
IF NOT NAMEP :X [STOP]
IF NUMBERP THING :X [MAKE :X 1 + THING →
:X]
END
```

(Note the use of **MAKE :X** rather than **MAKE "X**. It is not **X** that is being incremented. The value of **X** is not a number, but the name of another variable. It is that second variable that is incremented.)

```
?MAKE "TOTAL 7
?PRINT :TOTAL
7
?INC "TOTAL
?PRINT :TOTAL
8
?INC "TOTAL
?PRINT :TOTAL
9
```

For other examples, see the **MAKE** command.

THROW

Command

Format: **THROW** *name*

Remarks: This command is meaningful only within the range of a **CATCH** *name* command. See the **CATCH** command. If no **CATCH** is found with the corresponding input *name*, Logo prints the error message:

CAN'T FIND CATCH FOR *name*

Examples: In the following procedures, **THROW** “**TOPLEVEL**” returns control to top level. **TOPLEVEL** is a special word, denoting the mode in which commands can be executed directly without being embedded in a program. Compare with the effect of using **STOP**. Try the procedures, replacing **THROW** “**TOPLEVEL**” with **STOP**.

```
TO DRIVE
IF KEYP [LISTEN]
FD 2
DRIVE
END
```

```
TO LISTEN
MAKE "KEY READCHAR
IF :KEY = "S [THROW "TOPLEVEL]
IF :KEY = "R [RT 10]
IF :KEY = "L [LT 10]
END
```

```
?DRIVE
```

Press S, and the procedure stops, returning control to top level, not to **DRIVE**.

Format: TO *name*
 TO *name input1 input2 . . .*

Remarks: TO tells Logo that you are defining a procedure called *name*, with inputs (if any) as indicated. You do not have to place a quotation before *name*, *input1* or *input2*. TO does this for you. The prompt changes from “?” to “>” to remind you that you are defining a procedure. Logo does not carry out the actions that you type; it makes them part of the procedure definition. The special word END must be used alone on the last line of the definition to stop defining the procedure and return Logo to top level. These are acceptable inputs for TO;

TO ACCEPT

The name of the procedure is not quoted.

TO "ACCEPT

The name is quoted.

TO :ACCEPT

The : (colon) before the name is not evaluated.

TO does not accept a number, a primitive name, or the name of an already defined procedure, as its input.

TO Command

Examples: ?TO SQUARE :SIDE

>FD :SIDE

>RT 90

>FD :SIDE

>RT 90

>FD :SIDE

>RT 90

>FD :SIDE

>RT 90

>END

SQUARE DEFINED

?SQUARE 50

?TO ANNOUNCE :FIRSTNAME :LASTNAME

>PRINT [WE'RE HAPPY TO ANNOUNCE THE BIRTH OF]

>PRINT (SE :FIRSTNAME "C. :LASTNAME)

>PRINT [9 POUNDS 7 OUNCES]

>END

ANNOUNCE DEFINED

?ANNOUNCE "ANTHONY "PECORELLA

WE'RE HAPPY TO ANNOUNCE THE BIRTH OF

ANTHONY C. PECORELLA

9 POUNDS 7 OUNCES

TONE Command

Format: TONE *freq duration*

Remarks: Produces a tone. The *freq* input is the desired frequency in Hertz (cycles per second). The *freq* must be a number in the range of 37 through 19723. The *duration* input is the desired duration in clock ticks. The clock ticks occur 18.2 times per second. The *duration* must be a number in the range of 0 to 9999. TONE rounds both *freq* and *duration* to the nearest integer.

When TONE produces a sound, the sound continues until you give another TONE command. If the *duration* of the new TONE command is 0, the current sound is turned off. Otherwise, the first TONE must be completed before executing the second TONE. You can interrupt the second TONE by pressing Ctrl-Break, but not the first.

TONE

Command

The tuning note, A, has a frequency of 440. The following table correlates notes with their frequency for two octaves on either side of middle C.

Note	Frequency	Note	Frequency
C	130.810	C*	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
B	246.940	B	987.770
C	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
B	493.880	B	1975.500

*middle C. Higher (or lower) notes may be approximated by doubling (or halving) the frequency of the corresponding note in the previous (next) octave.

To create periods of silence, use TONE 19000
duration

The duration for one beat can be calculated from beats per minute by dividing the beats per minute into 1092 (the number of clock ticks in a minute).

TONE Command

The suggested minimum time for *duration* is three ticks.

The next table shows typical tempos in terms of clock ticks:

Tempo		Beats/ Minute	Ticks/ Beat
very slow ↓ slow ↓ medium ↓ fast ↓ very fast	Larghissimo		
	Largo	40-60	27.3-18.2
	Larghetto	60-66	18.2-16.55
	Grave		
	Lento		
	Adagio	66-76	16.55-14.37
	Adagietto		
	Andante	76-108	14.37-10.11
	Andantino		
	Moderato	108-120	10.11-9.1
	Allegretto		
	Allegro	120-168	9.1-6.5
	Vivace		
	Veloce		
	Presto	168-208	6.5-5.25
	Prestissimo		

TONE

Command

Examples: ?TONE 523.250 18.2

produces the middle C sound for one second.

```
TO SIREN :FREQ  
IF :FREQ > 440 [STOP]  
TONE :FREQ 3  
SIREN :FREQ + 5  
TONE :FREQ 3  
END
```

?SIREN 37

The SIREN procedure produces a siren sound by ascending and descending notes.

Format: TOWARDS *position*

Remarks: Outputs the heading the turtle would have if it were facing *position*.



FD 40



SETHEADING TOWARDS [20 10]

SETHEADING TOWARDS [20 10] heads the turtle in the direction of the position [20 10].

Examples: The procedure RANDOMBOX places a box at a random position on the screen. It uses MAKE to remember that position.

```
TO RANDOMBOX
PENUP
RIGHT RANDOM 360
FORWARD RANDOM 100
MAKE "PLACE POS
PENDOWN
SETHEADING 0
REPEAT 4 [FORWARD 10 RIGHT 90]
PENUP HOME
END
```

?RANDOMBOX

If you are playing a target game, you can use TOWARDS to find the direction the turtle should be facing.

```
?PRINT TOWARDS :PLACE
35
```

TYPE

Command

Format: *TYPE object*
 (*TYPE object1 object2 . . .*)

Remarks: Prints its inputs on the screen. TYPE does not issue a carriage return, so the prompt and cursor appear at the end of the message it printed. The outermost brackets of lists are not printed. Compare with the PRINT and SHOW commands.

Examples: ?TYPE "A
 A?TYPE "A TYPE [A B C]
 AA B C?(TYPE "A [A B C])
 AA B C?

The procedure PROMPT types a message followed by a space.

```
TO PROMPT :MESSAGE
TYPE :MESSAGE
TYPE CHAR 32
END
```


The procedure **MOVE** calls **PROMPT** to move the turtle interactively.

```
TO MOVE  
  PROMPT [HOW MANY STEPS SHOULD I TAKE?]  
  FD READWORD  
MOVE  
END
```

```
?MOVE  
HOW MANY STEPS SHOULD I TAKE? 50  
HOW MANY STEPS SHOULD I TAKE? 37  
HOW MANY STEPS SHOULD I TAKE? 2  
HOW MANY STEPS SHOULD I TAKE? 108
```

To stop, press Ctrl-Break.

UNBURY

Command

Format: UNBURY *package*

Remarks: Unburies all procedures and names in *package* so they are visible in your workspace. See the BURY command. Thereafter, when commands such as POALL, PONS, POPS, or POTS are used without an input, the unburi ed procedures or names will be included in what is printed on the screen.

Examples: ?POTS
TO LENGTH :OBJ
TO GREET
?UNBURY "SHAPES

?POTS
TO POLY :SIDE :ANGLE
TO LENGTH :OBJ
TO GREET
TO SPI :SIDE :ANGLE :INC

WAIT Command

Format: WAIT n

Remarks: Tells Logo to wait for n . The n is the amount of time in clock ticks. Clock ticks occur 18.2 times per second. The number of clock ticks in a minute is 1092. WAIT rounds n to the nearest integer. The maximum value of n is 9999.

Examples: The procedure REPORT keeps printing the turtle's position as it moves randomly. It uses WAIT to give you some time to read the position.

```
TO REPORT
RT 10 * RANDOM 36
FD 10 * RANDOM 10
PR POS
WAIT 100
REPORT
END
```

```
?CS HT
?REPORT
```

The following procedure cycles through all background colors on the graphics screen.

```
TO CHANGEBG
IF BG = 15 [SETBG 0]
SETBG 1 + BG
WAIT 30
CHANGEBG
END
```

WIDTH

Operation

Format: WIDTH

Remarks: Outputs the current screen width, a number from 2 through 80. When Logo starts up, the screen width is 40 for a monitor or TV and 80 for an IBM Monochrome Display.

You cannot have graphics with a screen width over 40, unless you are working with two screens (.SCREEN is 2). See the SETWIDTH command to change the width of the screen.

Example: ?PRINT WIDTH
40

WINDOW Command

Format: WINDOW

Remarks: Removes the boundaries from the turtle field. What you see is a portion of the turtle field as if looking through a small window. When the turtle moves beyond the visible bounds of the screen, it continues to move but can't be seen.

Initially, the screen is 250 turtle steps high, depending on the scrunch factor, and 320 steps wide. The entire turtle field is 9999 steps high and 9999 steps wide. See the FENCE and WRAP commands.

Example: ?WINDOW
?CS RT 5
?FD 500
?PRINT POS
43.579 498.093

WORD

Operation

Format: **WORD** *word1 word2*
 (**WORD** *word1 word2 word3 . . .*)
 (**WORD** *word1*)

Remarks: Outputs a word made up of its inputs. If **WORD** has one input, or more than two inputs, you must enclose **WORD** and its inputs in parentheses.

Examples: ?PRINT WORD "SUN "SHINE
 SUNSHINE

```
?PRINT (WORD "CHEESE "BURG "ER)
CHEESEBURGER
```

```
?PRINT WORD "BURG [ER]
WORD DOESN'T LIKE [ER] AS INPUT
```

The procedure **SUFFIX** puts **AY** at the end of its input.

```
TO SUFFIX :WD
  OUTPUT WORD :WD "AY
END
```

```
?PRINT SUFFIX "ANTEATER
ANTEATERAY
```

WORD Operation

The essence of the procedure SUFFIX is incorporated into PIG and LATIN, which translate a sentence into a dialect of Pig Latin.

```
TO LATIN :SENT
IF EMPTY P :SENT [OP [ ]]
OP SE PIG FIRST :SENT LATIN BF :SENT
END

TO PIG :WORD
IF MEMBER P FIRST :WORD [A E I O U Y] [O→
P WORD :WORD "AY]
OP PIG WORD BF :WORD FIRST :WORD
END

?PRINT LATIN [NO PIGS HAVE EVER SPOKEN →
PIG LATIN AMONG HUMANS]
ONAY IGSPAY AVEHAY EVERAY OKENSPAY IGPA→
Y ATINLAY AMONGAY UMANSAY
```

Note that LATIN does not work with words having no vowels. Press Ctrl-Break in this situation.

WORDP

Operation

Format: WORD *object*

Remarks: Outputs TRUE if *object* is a word; otherwise, FALSE.

Examples: ?PRINT WORDP "ZAM
TRUE

?PRINT WORDP 3
TRUE

?PRINT WORDP [3]
FALSE

?PRINT WORDP [E GRESS]
FALSE

?PRINT WORDP []
FALSE

?PRINT WORDP "
TRUE

?PRINT WORDP BUTFIRST "BURG
TRUE

?PRINT WORDP BUTFIRST [BURG]
FALSE

The following procedure decides if its input is a word or a list. If the input is a word, CHECK determines if it is a number.

```
TO CHECK :INPUT
TEST WORDP :INPUT
IFTRUE [IF NUMBER :INPUT [PR [NUMBER]] →
[PR [WORD]]]
IFFALSE [PR [LIST]]
END
```

```
?CHECK 5
NUMBER
```

```
?CHECK "FIVE
WORD
```

```
?CHECK [IAN MACMILLAN]
LIST
```

WRAP

Command

Format: WRAP

Remarks: Makes the turtle field wrap around the edges of the screen. If the turtle moves beyond one edge of the screen, it continues from the opposite edge. The turtle never leaves the visible bounds of the screen; when it tries to, it “wraps around” to the other side. See the FENCE and WINDOW commands.

Examples: ?WRAP
?CS RT 5
?FD 500
?PRINT POS
43.579 -1.907

WRITEOFF

Operation

Format: WRITEOFF

Remarks: Stands for “WRITE End-Of-File”.

Outputs **TRUE** if the write pointer is at the end of the file that is being currently written; otherwise **FALSE**. You must have previously opened a file for writing before using **WRITEOFF**. If you have not, Logo prints the error message:

NO FILE SELECTED

Notice that the commands **PRINT**, **TYPE**, and **SHOW** are directed to the file or device when you set the writing device. In other words, if you type

PRINT WRITEOFF

to see if you are at the end-of-file position, you won't see the result on the screen. Instead, **TRUE** will be written into the file for which you had used **SETWRITE**. See Chapter 4, “File Handling,” for more information on file handling.

Example: **?OPEN "ACCOUNTS**
?SETWRITE "ACCOUNTS
?WRITEOFF
I DON'T KNOW WHAT TO DO WITH TRUE

The command **SETWRITE** puts the write pointer at the end-of-file position.

WRITEPOS

Command

Format: WRITEPOS

Remarks: Stands for “WRITE POSition”.

Outputs the position of the write pointer in the file currently being written. To set the write position, see the SETWRITEPOS command. You must open a file for writing before using WRITEPOS or Logo prints the error message:

NO FILE SELECTED

If you try WRITEPOS for a device that can't be positioned, the error message

CAN'T POSITION DEVICE

is printed on the screen. For more information, see Chapter 4, “File Handling.”

WRITEPOS Command

Examples: ?OPEN "TELNOS
?SETWRITE "TELNOS
?WRITEPOS
I DON'T KNOW WHAT TO DO WITH 33

The file position is 33.

You cannot use PRINT WRITEPOS to find out the current write position because this would print 33 into the file TELNOS.

CHECKPOS is a procedure that prints the file position of the file given as input.

```
TO CHECKPOS :FILE
MAKE "POS WRITEPOS
SETWRITE "CON
PR :POS
SETWRITE :FILE
SETWRITEPOS :POS
END
```

```
?CHECKPOS "TELNOS
33
```

WRITER

Operation

Format: WRITER

Remarks: Outputs the current file or device that is open for writing. If the file is on the diskette in the default drive, then the drive number is not included in the output. Compare this with the ALLOPEN operation. See “Data Files” in Chapter 4 for information.

Examples: TO CHECKWRITE :FILE :DATA
IF NOT EQUALP WRITER :FILE [OPEN :FILE →
SETWRITE :FILE]
PRINT :DATA
CLOSE :FILE
END

?CHECKWRITE "CLASS.LIS [KIYOKO OKUMURA]

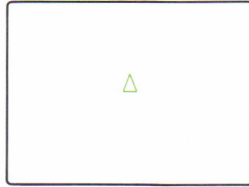
The CHECKWRITE procedure first decides if a file is open for writing. If it is not, CHECKWRITE opens the file for writing. It then sends data to that file before closing it.

XCOR Operation

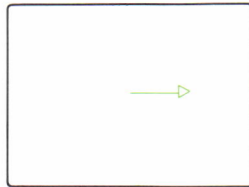
Format: XCOR

Remarks: Outputs the x-coordinate of the current position of the turtle.

Examples:



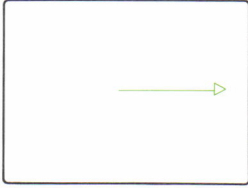
```
?CS  
?PRINT XCOR  
0
```



```
?RT 90 FD 50  
PRINT XCOR  
50
```

XCOR

Operation



```
?SETX 2 * XCOR  
?PR XCOR  
100
```

moves the turtle horizontally to a position twice as far from the y-axis as it used to be.

The following procedure outputs the cosine value of an angle.

```
TO COSINE :ANGLE  
HOME  
SETHEADING 90  
LEFT :ANGLE  
FORWARD 100  
OUTPUT XCOR / 100  
END
```

```
?PRINT COSINE 30  
0.86601
```


YCOR Operation

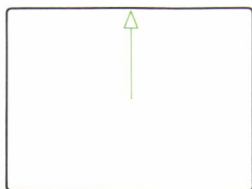
Format: YCOR

Remarks: Outputs the y-coordinate of the current position of the turtle.

Examples:



```
?CS  
?PRINT YCOR  
0
```



```
?FD 70  
?PRINT YCOR  
70
```

YCOR

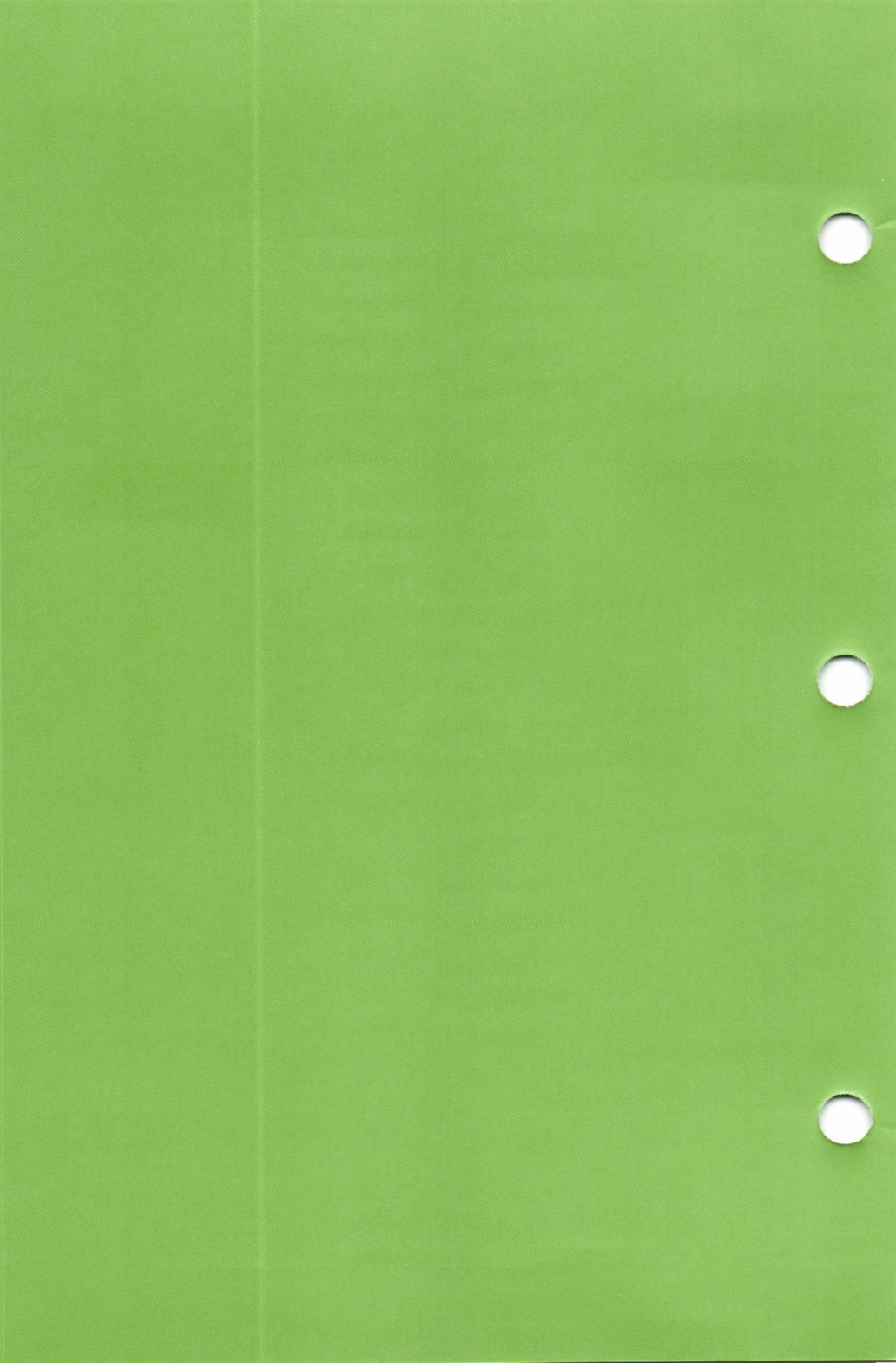
Operation

The following procedure outputs the sine value of an angle.

```
TO SINE :ANGLE  
HOME  
SETHEADING 90  
LEFT :ANGLE  
FORWARD 100  
OUTPUT YCOR / 100  
END
```

```
?PRINT SINE 30  
0.5
```

Part 3. Appendixes



Appendix A. Messages

As you work with Logo, you will cause different messages to be printed on the screen. Most of these messages are error messages, which occur when Logo detects an error. Other messages give you information such as the amount of procedures you saved in a file.

This appendix contains three lists of messages. The first is a list of information messages with an explanation of when they occur. The second is a list of error messages organized alphabetically. This list includes information about the message, along with its associated error message number. The third is a list of error messages organized by number. Note that the *name* in the message is the name of a primitive, procedure, or variable. It will be filled in when Logo prints the message.

Information Messages

***name* DEFINED**

Printed after exiting from the Logo editor by pressing the Esc key. It means that the procedure *name* is defined and is in your workspace.

***n* PROCEDURES SAVED**

Logo prints the amount of procedures saved after you have used the SAVE command.

***name* SKIPPED**

Caused by giving a procedure a primitive name. The message is printed when you leave the Logo editor, or when you load a file containing a procedure whose name is a primitive.

<u>Error Message</u>	<u>Number</u>
ABSOLUTELY OUT OF SPACE	22
<p>The absolute limit of the workspace has been reached. If this ever happens, contact your IBM Personal Computer dealer.</p>	
CAN BE USED ONLY IN A PROCEDURE	33
<p>Some primitives, such as LOCAL, can be used only within a procedure.</p>	
<i>name</i> CAN'T BE USED IN A PROCEDURE	5
<p>Certain primitives cannot be used within a procedure. For example, CO (COninue) cannot be used within a procedure, although PAUSE can be.</p>	
CAN'T CLOSE A DRIBBLE FILE	42
<p>A dribble file can be closed only with the command NODRIBBLE.</p> <p>For example:</p> <pre>?DRIBBLE "LPT1 ?CLOSE "LPT1 CAN'T CLOSE A DRIBBLE FILE</pre>	

<u>Error Message</u>	<u>Number</u>
CAN'T DO GRAPHICS IN THIS MODE	3
<p>A graphics command was given with a monochrome display screen. An IBM Color Graphics/Monitor Adapter and a monitor or TV is required with any graphics commands.</p> <p>The message also appears if the screen has a width of over 40 columns. To overcome the problem enter SETWIDTH 40.</p>	
CAN'T DIVIDE BY ZERO	13
<p>A command was given to divide a number by zero, or division by zero occurred during a calculation.</p>	
CAN'T DRIBBLE ON SCREEN	18
<p>The screen (CON) was given as an input to the command DRIBBLE.</p>	
CAN'T FIND CATCH FOR <i>name</i>	21
<p>THROW <i>name</i> was used in a set of procedures without CATCH <i>name</i>.</p>	
CAN'T FIND LABEL <i>name</i>	7
<p>GO <i>name</i> was used in a procedure without LABEL <i>name</i>.</p> <p>To fix the procedure, insert LABEL <i>name</i> in the appropriate place.</p>	

Error MessageNumber**CAN'T OPEN DEVICE**

45

Some devices such as CON (the screen) can't be used as an input to OPEN, EDITFILE, or POFILE.

CAN'T POSITION DEVICE

32

The commands SETREADPOS or SETWRITEPOS can't be used for the printer or the serial communications line.

CAN'T READ DEVICE

24

A device, such as a printer or serial communication line, given as input to POFILE, EDITFILE, SETREAD, or LOAD cannot be read.

CAN'T USE *name* WHILE LOADING

49

The primitive LOAD was used with a file containing a file command or an editor command.

For example,

```
?EDITFILE "ABC
```

```
DRIBBLE "FILE.TRY  
TO TRY  
PR [HELLO GOODBYE]  
END
```

```
?LOAD "ABC.  
CAN'T USE DRIBBLE WHILE LOADING
```

This error message only occurs when the file command or editor command is not within a procedure.

Following are some of the commands which cause this error: OPEN, LOAD, DIR, FILEP, DRIBBLE, EDITFILE, EDIT, EDNS, ALLOPEN, POFILE, FILELEN, SETREAD, SETWRITE.

DEVICE UNAVAILABLE

39

A system device is not attached to the computer.

You will get this message, for example, if the Asynchronous Communications Adapter is not attached to the computer, and you type .SETCOM 1 1200 0 8 1.

name* DIDN'T OUTPUT TO *name

10

An input was not given to a procedure or a primitive that required one and that was followed on the same line by another procedure or primitive.

For example:

```
?PO PO "LOOP  
TO LOOP  
LABEL "NUM  
PRINT RANDOM 100  
GO "NUM  
END
```

PO DIDN'T OUTPUT TO PO

The first PO expects an input; it got PO as its input. PO is a command, not an operation. Therefore, it didn't output to the first PO.

Error Message

Number

?RT CS

produces the error message

CLEARSCREEN DIDN'T OUTPUT TO RIGHT

RT expects an input. CS is a command; it doesn't output to RT.

DISK FULL

12

All sectors on the disk are used and more files cannot be created, or you have more than 64 files in the diskette root directory. Erase some files or use a new disk. When you receive this message, the opened file will automatically be closed.

name DOESN'T LIKE *name* AS INPUT

41

An incorrect input was given to a procedure (or primitive).

For example:

?PR 1 +)

gives the error message:

+ DOESN'T LIKE) AS INPUT

?PR LAST POS - 7

- DOESN'T LIKE [0 0] AS INPUT

To fix this, put LAST POS in parentheses: (LAST POS) - 7

<u>Error Message</u>	<u>Number</u>
<i>name</i> DOESN'T LIKE PRECISION <i>name</i>	48
Certain arithmetic primitives, such as SIN, COS, and ARCTAN, will not run with PRECISION greater than 100. Check PRECISION and lower it with SETPRECISION.	
DRIBBLE FILE ALREADY SELECTED	43
DRIBBLE was used with a file before the current dribble file was closed with NODRIBBLE. Only one dribble file can be open at a time.	
EDITOR BUFFER IS FULL	8
The input to EDITFILE or EDIT is too large for the editor buffer, which has a capacity of 4096 bytes.	

**ERROR ON COMMUNICATION
LINE**

46

An error occurred while sending a character on the serial communication line.

This message occurs for example when:

- there is no device plugged into the Asynchronous Communications Adapter.
- the device plugged into the communications adapter is not switched on.
- the inputs of .SETCOM are not valid.

FILE *name* ALREADY EXISTS

15

A file that already exists cannot be saved. This error also occurs when the second input to EDITFILE already exists. Use another file name.

<u>Error Message</u>	<u>Number</u>
FILE <i>name</i> DOES NOT EXIST	17
<p>A file that does not exist can't be loaded or erased.</p> <p>For example:</p> <pre>?LOAD "DATA FILE DATA DOES NOT EXIST ?LOAD "DATA.</pre> <p>If the file does not have an extension, LOAD and ERASEFILE require a period following the file name.</p> <p>POFILE, EDITFILE, and OPEN require the entire file specification. See “Naming Files” in Chapter 4.</p>	
FILE <i>name</i> ALREADY OPEN	16
<p>If a file is opened by the command OPEN or selected as DRIBBLE, it cannot be opened, dribbled into, saved to, erased or loaded. First close the file with CLOSE “<i>name</i> or NODRIBBLE.</p>	
FILE <i>name</i> ALREADY SELECTED FOR DRIBBLE	44
<p>A file selected for DRIBBLE can't be used as an input to SETWRITE or SETREAD. First close the file with NODRIBBLE.</p>	
FILE <i>name</i> IS NOT OPEN	14
<p>A file that is not open can't be used as an input to SETREAD, SETWRITE, FILELEN, or CLOSE.</p>	

Error Message

Number

name HAS NO VALUE

36

A variable was used, but it was not defined.

For example:

```
TO TRI
PR :WD
IF EMPTY :WD [STOP]
TRI BUTFIRST :WD
END
```

```
?TRI
```

gives the error message

```
WD HAS NO VALUE IN TRI:
PR :WD
```

I DON'T KNOW HOW TO *name*

35

Logo thinks *name* is a procedure, tries to execute it, but can't find its definition.

For example:

```
?PR MARY
I DON'T KNOW HOW TO MARY
?HELLO
I DON'T KNOW HOW TO HELLO
```

<u>Error Message</u>	<u>Number</u>
I DON'T KNOW WHAT TO DO WITH <i>name</i>	38

An input was given without specifying its procedure or primitive. This message also occurs when an operation is entered without a command immediately preceding it on a line.

For example:

```
?PRINT "JOHN "MARY  
JOHN  
I DON'T KNOW WHAT TO DO WITH MARY
```

The way PRINT is written here, it takes a single input.

```
?BUTFIRST "LOGO  
I DON'T KNOW WHAT TO DO WITH OGO
```

If PRINT is inserted as the first word in the above example, no error message will occur.

I DON'T UNDERSTAND YOUR FILE SPECIFICATION	47
---	----

The file name is not a valid file specification. See "Naming Files" in Chapter 4.

For example:

```
?SAVE "Z:DEMO  
I DON'T UNDERSTAND YOUR FILE  
SPECIFICATION
```

There is no drive Z on your computer. Change Z to A, B, or the drive wanted.

<u>Error Message</u>	<u>Number</u>
I'M HAVING TROUBLE WITH THE DISK	11
<p>A primitive or procedure was used that directly accesses information on disk (such as DIR, ERASEFILE, LOAD, SAVE). This could be the result of an error on the disk. If a diskette is being used, the drive door may not be closed, there may not be a diskette in the drive, the diskette may not be formatted, or the diskette may have a write-protect tab.</p>	
I'M HAVING TROUBLE WITH THE PRINTER	40
<p>The printer (LPT1) was given as an input even though the printer is not on, a Printer Adapter is not in the computer, or the printer ran out of paper.</p>	
<i>name</i> IS A PRIMITIVE	6
<p>The name of a primitive was given as the input to TO, PO, or ERASE. For example, FIRST was used as the name of a procedure. In this case, give the procedure another name.</p>	
<i>name</i> IS ALREADY DEFINED	1
<p>The name given to define a procedure has already been used as a procedure name.</p> <p>Give the new procedure another name, or save the old procedure in a file and erase it from your workspace before defining the new procedure.</p>	

<u>Error Message</u>	<u>Number</u>
<i>name</i> IS NOT TRUE OR FALSE	25
<p>An input was given to IF, AND, OR, NOT, or SETCAPS that was not a predicate; that is, it did not output TRUE or FALSE.</p> <p>For example:</p> <pre> TO CHECKOR IF OR 1 [OUTPUT "TRUE] END </pre> <p>The input to OR, which is 1, is not a predicate.</p>	
<i>name</i> IS UNDEFINED	9
<p>The command PO (Print Out) or ERASE has been given an input not yet defined as a procedure.</p>	
LOGO SYSTEM BUG	50
<p>If this message is displayed contact your IBM Personal Computer dealer.</p>	
NO FILE SELECTED	4
<p>SETWRITEPOS, SETREADPOS, WRITEPOS, or READPOS was used before a file was selected with SETWRITE or SETREAD.</p>	

<u>Error Message</u>	<u>Number</u>
<p>NOT ENOUGH INPUTS TO <i>name</i></p> <p>Not enough inputs were given to a procedure or primitive that is being run.</p> <p>For example:</p> <pre>?PR BUTFIRST</pre> <p>gives the error message</p> <pre>NOT ENOUGH INPUTS TO BUTFIRST</pre> <pre>?FD</pre> <p>gives the error message</p> <pre>NOT ENOUGH INPUTS TO FORWARD</pre>	29
<p>NUMBER TOO BIG</p> <p>Inputs to mathematical operations have more than 1000 significant digits. Or, in the course of performing mathematical operations, a number with an exponent greater than 9999 is generated, Logo prints this message.</p>	2
<p>ONLY 5 FILES CAN BE OPEN</p> <p>A file was opened when five files are already open, including a DRIBBLE file. First close a file before opening a new one.</p>	20

<u>Error Message</u>	<u>Number</u>
OUT OF SPACE	23
<p>Your workspace is completely filled. Procedures can't be written or edited, loaded, or saved.</p> <p>Try to erase some procedures and names from your workspace, then type RECYCLE. If this doesn't work, restart Logo.</p>	
PAUSING...	26
<p>You pressed the function key F5 or typed PAUSE, thereby putting Logo into a PAUSE mode. To get back into the normal mode, type CO.</p>	
POSITION OUT OF RANGE	37
<p>An input beyond the end-of-file position was given to SETREADPOS or SETWRITEPOS.</p>	
STOPPED!	28
<p>You pressed Ctrl-Break, which interrupts whatever is running.</p>	
TOO FEW ITEMS IN <i>name</i>	19
<p>The first input to ITEM was too large for the amount of elements in the second input to ITEM.</p> <p>For example:</p> <pre>?PR ITEM 6 [A B] TOO FEW ITEMS IN [A B]</pre>	

<u>Error Message</u>	<u>Number</u>
<p>TOO MANY INPUTS TO <i>name</i></p> <p>Too many inputs were given to a primitive or a procedure.</p> <p>For example:</p> <pre>?(MAKE "A "B "C)</pre> <p>gives the error message</p> <pre>TOO MANY INPUTS TO MAKE</pre>	30
<p>TOO MUCH INSIDE () 'S</p> <p>Parentheses were incorrectly placed in a Logo instruction.</p> <p>For example:</p> <pre>?PR SUM (5 (6 * 3))</pre> <p>will produce this error message, but PR (SUM 5 (6 *3)) will not.</p>	31

<u>Error Message</u>	<u>Number</u>
TURTLE OUT OF BOUNDS	34
<p>The command FENCE was previously used in the graphics mode. Then, an input to FORWARD, BACK, DOT, SETX, SETY or SETPOS was given that would put the turtle off the screen.</p> <p>For example:</p> <pre>?FENCE ?FD 300 TURTLE OUT OF BOUNDS</pre>	

YOU'RE AT TOPLEVEL	27
<p>Certain commands (for example, STOP, CO, and OUTPUT) can only be used in a procedure.</p>	

Error Message Numbers	Message
1	<i>name</i> IS ALREADY DEFINED
2	NUMBER TOO BIG
3	CAN'T DO GRAPHICS IN THIS MODE
4	NO FILE SELECTED
5	<i>name</i> CAN'T BE USED IN A PROCEDURE
6	<i>name</i> IS A PRIMITIVE
7	CAN'T FIND LABEL <i>name</i>
8	EDITOR BUFFER IS FULL
9	<i>name</i> IS UNDEFINED
10	<i>name</i> DIDN'T OUTPUT TO <i>name</i>
11	I'M HAVING TROUBLE WITH THE DISK
12	DISK FULL
13	CAN'T DIVIDE BY ZERO
14	FILE <i>name</i> IS NOT OPEN
15	FILE <i>name</i> ALREADY EXISTS
16	FILE <i>name</i> ALREADY OPEN
17	FILE <i>name</i> DOES NOT EXIST
18	CAN'T DRIBBLE ON SCREEN
19	TOO FEW ITEMS IN <i>name</i>
20	ONLY 5 FILES CAN BE OPEN
21	CAN'T FIND CATCH FOR <i>name</i>
22	ABSOLUTELY OUT OF SPACE
23	OUT OF SPACE
24	CAN'T READ DEVICE
25	<i>name</i> IS NOT TRUE OR FALSE
26	PAUSING ...
27	YOU'RE AT TOPLEVEL
28	STOPPED!
29	NOT ENOUGH INPUTS TO <i>name</i>
30	TOO MANY INPUTS TO <i>name</i>
31	TOO MUCH INSIDE ()'S
32	CAN'T POSITION DEVICE
33	CAN BE USED ONLY IN A PROCEDURE
34	TURTLE OUT OF BOUNDS
35	I DON'T KNOW HOW TO <i>name</i>
36	<i>name</i> HAS NO VALUE
37	POSITION OUT OF RANGE
38	I DON'T KNOW WHAT TO DO WITH <i>name</i>
39	DEVICE UNAVAILABLE
40	I'M HAVING TROUBLE WITH THE PRINTER
41	<i>name</i> DOESN'T LIKE <i>name</i> AS INPUT
42	CAN'T CLOSE A DRIBBLE FILE
43	DRIBBLE FILE ALREADY SELECTED
44	FILE <i>name</i> ALREADY SELECTED FOR DRIBBLE
45	CAN'T OPEN DEVICE
46	ERROR ON COMMUNICATION LINE
47	I DON'T UNDERSTAND YOUR FILE SPECIFICATION
48	<i>name</i> DOESN'T LIKE PRECISION <i>name</i>
49	CAN'T USE <i>name</i> WHILE LOADING
50	LOGO SYSTEM BUG

Appendix B. Useful Tools

The procedures presented here alphabetically are for your convenience when constructing your own procedures. Some of them appear in Chapter 7, “Logo Primitives” (see Index) and others appear here for the first time. These procedures are on the Logo Language Diskette in the file “TOOLS.LF”.

; will allow the use of the rest of a line for comments in a procedure.

```
TO ; :COMMENTS  
END
```

If there is more than one word, use brackets.

For example, see the procedure **ABS :NUM** below.

ABS outputs the absolute value of its input.

```
TO ABS :NUM  
; [THE OUTPUT OF ABS IS]  
; [ALWAYS POSITIVE]  
OP IF :NUM < 0 [-:NUM] [:NUM]  
END
```

ARCL and **ARCR** draw left and right turn arcs, respectively. Their inputs are :RADIUS, the radius of the circle from which the arc is taken, and :DEGREES, the degrees of the arc (the length of the edge).

```
TO ARCL :RADIUS :DEGREES
LOCAL "STEP LOCAL "REM
MAKE "STEP 2 * :RADIUS * 3.1416 / 36
MAKE "REM REMAINDER :DEGREES 10
REPEAT :DEGREES / 10 [LT 5 FD :STEP LT →
5]
IF :REM > 0 [FD :STEP * :REM / 10 LT :R→
EM]
END
```

```
TO ARCR :RADIUS :DEGREES
LOCAL "STEP LOCAL "REM
MAKE "STEP 2 * :RADIUS * 3.1416 / 36
MAKE "REM REMAINDER :DEGREES 10
REPEAT :DEGREES / 10 [RT 5 FD :STEP RT →
5]
IF :REM > 0 [FD :STEP * :REM / 10 RT :R→
EM]
END
```

CIRCLEL and **CIRCLER** draw left and right turn circles with a specified radius as input.

```
TO CIRCLEL :RADIUS
LOCAL "STEP
MAKE "STEP 2 * :RADIUS * 3.1416 / 36
REPEAT 36 [LT 5 FD :STEP LT 5]
END
```

```
TO CIRCLER :RADIUS
LOCAL "STEP
MAKE "STEP 2 * :RADIUS * 3.1416 / 36
REPEAT 36 [RT 5 FD :STEP RT 5]
END
```


CONVERT converts N, a number, from a base value (:FRBASE) to another base value (:TOBASE).

```
TO CONVERT :N :FRBASE :TOBASE
OP DEC.TO.ANYBASE ANYBASE.TO.DEC :N :FR→
BASE 1 :TOBASE
END
```

```
TO ANYBASE.TO.DEC :N :BASE :POWER
IF EMPTY :N [OP 0]
OP (:POWER * C.TO.N LAST :N) + ANYBASE.→
TO.DEC BL :N :BASE :POWER * :BASE
END
```

```
TO DEC.TO.ANYBASE :N :BASE
IF :N < :BASE [OP N.TO.C :N]
OP WORD DEC.TO.ANYBASE INT QUOTIENT :N →
:BASE :BASE N.TO.C REMAINDER :N :BASE
END
```

```
TO C.TO.N :N
IF NUMBERP :N [OP :N]
OP (ASCII :N) - 55
END
```

```
TO N.TO.C :N
IF :N < 10 [OP :N]
OP CHAR 55 + :N
END
```

You then can use **CONVERT** to convert decimal to hexadecimal or hexadecimal to decimal.

```
TO DECTOHEX :N
OP CONVERT :N 10 16
END
```

```
TO HEXTODEC :N
OP CONVERT :N 16 10
END
```

DIVISORP indicates (TRUE or FALSE) whether its first input divides evenly into its second.

```
TO DIVISORP :A :B
OP 0 = REMAINDER :B :A
END
```

DRIVE lets you drive the turtle around the screen with the touch of a key. This is an example of single-keypress interactive programming.

```
TO DRIVE
IF KEYP [LISTEN]
FD 1
DRIVE
END
```

```
TO LISTEN
MAKE "ANS RC
IF :ANS = "S [THROW "TOPLEVEL]
IF :ANS = "R [RT 10]
IF :ANS = "L [LT 10]
END
```

FOREVER repeats a group of instructions until Ctrl-Break is pressed or the power is switched off.

```
TO FOREVER :INSTRUCTIONLIST
RUN :INSTRUCTIONLIST
FOREVER :INSTRUCTIONLIST
END
```

LABELPIC lets you label a picture on the graphics portion of the screen using the cursor.

```
TO LABELPIC :LABEL :POS
CT
LOCAL "CURSORPOS
MAKE "CURSORPOS CURSOR
SETCURSOR :POS
TYPE :LABEL
SETCURSOR :CURSORPOS
END
```

See also **TITLE**. Both **LABELPIC** and **TITLE** permit you to mix graphics and text on the graphics portion of the screen.

MAP applies a command to every element of a list.

```
TO MAP :CMD :LIST
IF EMPTY? :LIST [STOP]
RUN LIST :CMD WORD "" FIRST :LIST
MAP :CMD BF :LIST
END
```

PEL outputs a number (0 – 3) that represents the color of the display picture element at the position indicated by the list of two numbers [x y]. See the **PENCOLOR** and **POS** operations.

The specified position is rounded to the nearest scan line on the display based on the current value of **.SCRUNCH** and to the nearest pixel on that scan line. The value at the resulting memory position of the display adapter is output. A value of zero represents the current background color. The colors associated with the values 1 – 3 depend on the palette selected.

```

TO PEL :POS
OUTPUT ITEM (BYTEPOS (ROUND FIRST :POS)→
) 4PELS :POS
END

```

```

TO BYTEPOS :XPEL
IF :XPEL > 0 [OUTPUT 1 + REMAINDER (:XP→
EL - 1) 4] [OUTPUT 4 + REMAINDER :XPEL →
4]
END

```

```

TO 4PELS :POS
LOCAL "P
LOCAL "N
MAKE "P []
MAKE "N DISPBYTE :POS
REPEAT 4 [MAKE "P FPUT REMAINDER :N 4 :→
P MAKE "N INT QUOTIENT :N 4]
OUTPUT :P
END

```

```

TO DISPBYTE :POS
OUTPUT .EXAMINE -18432 OFFSET :POS
END

```

```

TO OFFSET :POS
LOCAL "SCAN
MAKE "SCAN 100 - ROUND ((LAST :POS) * .→
SCRUNCH)
IF OR OR FIRST :POS > 160 FIRST :POS < →
-159 OR :SCAN > 199 :SCAN < 0 [PR [PLEA→
SE ENTER A POSITION WHICH IS] PR [WITHI→
N THE VISIBLE LIMITS OF THE] PR [GRAPHI→
CS SCREEN.] THROW "TOPLEVEL]
OUTPUT (8192 * REMAINDER :SCAN 2) + (80→
* INT QUOTIENT :SCAN 2) + (INT QUOTIEN→
T (159 + ROUND (FIRST :POS)) 4)
END

```

These procedures can be used to set the **PENCOLOR** by name. Each procedure sets the palette and outputs the correct color.

```
TO GREEN  
SETPAL 0  
OP 1  
END
```

```
TO RED  
SETPAL 0  
OP 2  
END
```

```
TO BROWN  
SETPAL 0  
OP 3  
END
```

```
TO CYAN  
SETPAL 1  
OP 1  
END
```

```
TO MAGENTA  
SETPAL 1  
OP 2  
END
```

```
TO WHITE  
SETPAL 1  
OP 3  
END
```

For example, SETPC RED will change the palette to zero and set the pen to red.

POLY draws a polygon “forever”.

```
TO POLY :SIDE :ANGLE  
FD :SIDE  
RT :ANGLE  
POLY :SIDE :ANGLE  
END
```

PROMPT displays a message and leaves a space; any resulting input is accepted on the same line.

```
TO PROMPT :MESSAGE  
TYPE :MESSAGE  
TYPE CHAR 32  
END
```

CHAR 32 will output space character to **TYPE**.

STEP lets you run a procedure line by line. **UNSTEP** restores the original procedure definition. **STEP** is useful for debugging procedures, especially if you are an inexperienced user. If it or any other procedure seems hard to understand, **STEP** may be used to “walk you through”. Note that you can **STEP** “**STEP** but you can’t **STEP** “**STEPPER**.

When you get the ?? prompt, Logo waits for you to press any key. You usually will want to **UNSTEP** all your procedures before saving your workspace.

```

TO SHOWARGS :ARGLIST
IF EMPTY :ARGLIST [STOP]
MAKE "NEWDEF LPUT (LIST "PRINT "SENTENC→
E (LIST (FIRST :ARGLIST) "IS) (WORD " : →
FIRST :ARGLIST)) :NEWDEF
SHOWARGS BF :ARGLIST
END

```

```

TO SHOWLINES :INSTRUCTIONS
IF EMPTY :INSTRUCTIONS [STOP]
MAKE "NEWDEF LPUT (LIST "TYPE FIRST :IN→
STRUCTIONS) :NEWDEF
MAKE "NEWDEF LPUT [STEPPER] :NEWDEF
MAKE "NEWDEF LPUT FIRST :INSTRUCTIONS :→
NEWDEF
SHOWLINES BF :INSTRUCTIONS
END

```

```

TO STEPPER
TYPE "??
IGNORE READLIST
END

```

```

TO IGNORE :INPUT
END

```

```

TO UNSTEP :PROC
IF EMPTY :PROC [STOP]
IF LISTP :PROC [UNSTEP FIRST :PROC UNST→
EP BF :PROC STOP]
IF EMPTY TEXT WORD ". :PROC [PR SE :PR→
OC [NOT STEPPED.] STOP]
COPYDEF :PROC WORD ". :PROC
ERASE WORD ". :PROC
END

```

```

TO STEP :PROC
IF EMPTY :PROC [STOP]
IF LISTP :PROC [STEP FIRST :PROC STEP B→
F :PROC STOP]
IF PRIMITIVEP :PROC [PR SE [CAN'T STEP →
PRIMITIVE] :PROC STOP]
IF EMPTY TEXT :PROC [PR SE [NO PROCEDU→
RE NAMED] :PROC STOP]
COPYDEF WORD ". :PROC :PROC
MAKE "OLDDEF TEXT :PROC
MAKE "NEWDEF (LIST FIRST :OLDDEF)
MAKE "NEWDEF LPUT (LIST "PRINT (LIST "E→
NTERING :PROC)) :NEWDEF
SHOWARGS FIRST :OLDDEF
SHOWLINES BF :OLDDEF
DEFINE :PROC :NEWDEF
END

```

TEACH lets you define a procedure as you are running it line by line. By typing **END**, you finish defining the procedure. Entering **ERASE** removes the previous line from the definition in progress. This is especially useful when working with young children. **TEACH** is intended for beginning turtle graphics procedures only.

```

TO TEACH
LOCAL "THISLINE
DEFINE "PROGRAM [[]]
CLEARSCREEN PENDOWN
ADDLINES
IF NOT EMPTY BUTFIRST TEXT "PROGRAM [N→
AMEIT]
END

```

```

TO ADDLINES
TYPE [>?]
MAKE "THISLINE READLIST
IF :THISLINE = [END] [STOP]
ADDIT :THISLINE
ADDLINES
END

```



```
TO ADDIT :THISLINE
IF EMPTY :THISLINE [STOP]
TEST :THISLINE = [ERASE]
IFTRUE [WIPEOUT]
IFFALSE [RUNSTORE]
END
```

```
TO WIPEOUT
DEFINE "PROGRAM BUTLAST TEXT "PROGRAM
CLEARSCREEN
RUN [PROGRAM]
END
```

```
TO RUNSTORE
CATCH "ERROR [RUN :THISLINE]
LOCAL "GOOF
MAKE "GOOF ERROR
TEST EMPTY :GOOF
IFTRUE [DEFINE "PROGRAM LPUT :THISLINE →
TEXT "PROGRAM]
IFFALSE [PRINT FIRST BUTFIRST :GOOF]
END
```

```
TO NAMEIT
LOCAL "NAME
PRINT [What should I call this?]
MAKE "NAME READLIST
TEST EMPTY :NAME
IFTRUE [ERASE "PROGRAM STOP]
TEST DEFINEDP FIRST :NAME
IFTRUE [TRYAGAIN]
IFFALSE [COPY]
END
```

```
TO TRYAGAIN
PRINT SENTENCE FIRST :NAME [is already →
defined]
PRINT []
NAMEIT
END
```

```
TO COPY
COPYDEF FIRST :NAME "PROGRAM
PRINT SENTENCE FIRST :NAME [defined]
ERASE "PROGRAM
END
```

These procedures can be used to set the **TEXTCOLOR** by name.

```
TO BLACK.BLUE
OP [0 1]
END
```

```
TO BLACK.GREEN
OP [0 2]
END
```

```
TO BLACK.CYAN
OP [0 3]
END
```

```
TO BLACK.RED
OP [0 4]
END
```

```
TO BLACK.MAGENTA
OP [0 5]
END
```

```
TO BLACK.BROWN
OP [0 6]
END
```

```
TO BLACK.WHITE
OP [0 7]
END
```

For example **SETTC BLACK.RED** will result in black text on a red background. These procedures can be adapted to the color combinations you would like to use.

TITLE allows you to place a title on the graphics screen using the turtle.

```
TO TITLE :XC :YC :LST
MAKE "NUM 1
PU SETPOS SE :XC :YC
RT 90 ST
REPEAT COUNT :LST [WRITEWORD ITEM :NUM →
:LST MAKE "NUM :NUM + 1]
SETSHAPE "TURTLE
HT PD SETH 0
END
```

```
TO WRITEWORD :WD
IF EMPTY :WD [FD 5 STOP]
SETSHAPE ASCII FIRST :WD
STAMP FD 12
WRITEWORD BF :WD
END
```

See also **LABELPIC**. Both **TITLE** and **LABELPIC** permit you to mix graphics and text on the graphics portion of the screen.

TRAIL stamps the turtle shape at regular increments for the distance specified.

```
TO TRAIL :DIST :INC
REPEAT :DIST/:INC [STAMP PU FD :INC]
FD REMAINDER :DIST :INC
END
```

WHICH outputs which position an element has in its list. **WHICH** "C [A B C] outputs 3. This is a complement to **ITEM**.

```
TO WHICH :MEMBER :LIST
IF NOT MEMBERP :MEMBER :LIST [OP 0]
IF :MEMBER = FIRST :LIST [OP 1]
OP 1 + WHICH :MEMBER BF :LIST
END
```

WHILE repeats a group of instructions until :CONDITION becomes false.

```
TO WHILE :CONDITION :INSTRUCTIONLIST
TEST RUN :CONDITION
IFFALSE [STOP]
RUN :INSTRUCTIONLIST
WHILE :CONDITION :INSTRUCTIONLIST
END
```

Appendix C. How Logo Uses Memory

Logo procedures and variables take up space; Logo uses additional space to run procedures.

Some Logo users may wish to know how space is used in Logo and how to conserve it. In general, saving space is not something you should worry about. Instead, you should try to write procedures as clearly and elegantly as possible. However, because computer memories have a limited amount of space, this appendix discusses how space is allocated in Logo and how you can use less of it.

How Logo Allocates Memory

Space in Logo is allocated in *nodes*, each of which is five bytes long. All Logo objects and procedures are built out of nodes. The internal workings of Logo also use nodes. The interpreter knows which nodes are available for use. When there are no more free nodes, a special part of Logo called the *garbage collector* looks through all nodes and reclaims any node not being used.

For example, during execution of the following procedure

```
TO PRINT.SUM :N :M  
MAKE "N :N + :M  
PRINT :N  
END
```

For example,

```
PRINT.SUM 3 4
```

N is assigned to a new node that holds the value 7. After PRINT.SUM stops, the old value of N (if any) is restored and the 7 will be lost forever. The node containing the 7 can be reused, and it will be reclaimed as a free node the next time the garbage collector runs. The garbage collector runs automatically when necessary, but you can force it to run by using the Logo command RECYCLE. This may be useful just before running a procedure whose function may be disrupted by the short, but unpredictable, pause required to do the garbage collection.

The operation NODES outputs the number of free nodes. However, if you really want to find out how much space you have, you should do something similar to the following:

```
?RECYCLE PRINT NODES  
259
```

How Space is Used

Every Logo word used is stored only once: all occurrences of that word are actually *pointers* to the word. A word takes up two nodes plus one node for every two letters in its name and an additional node if this word has a property list. Some words are made up of pointers to already existing words. For instance, if you say WORD "SAN "FRANCISCO, the resulting word SANFRANCISCO actually is made up of a pointer to "SAN" and a pointer to "FRANCISCO".

A number, whether integer or decimal, is made up of a list of its digits taking a node for every four digits and one for its exponent. A list takes up one node for each element (plus the size of the element itself). You can get a rough estimate of the size of a procedure by taking the size of the list that would be output by TEXT. For example, the output of TEXT "POLY is

```
[[[SIDE ANGLE] [FD :SIDE RT :ANGLE] [POL→  
Y :SIDE :ANGLE]]
```

It has three elements.

Space-Saving Hints

- Rewrite the program using procedures to replace repetitive sections of the program.
- Don't create new words. The names of local variables of procedures can be the same as names of local variables of other procedures. The names of procedures and primitives can also be used as variable names. Remember that you can reuse or erase primitive names. Doing so does not destroy the word but does reclaim two nodes that were used to store internal information about the primitive.
- Misspellings, typing errors, and words that are no longer being used are not destroyed until you do a **RECYCLE**. All currently existing words can be seen on the list output by **.CONTENTS**. For instance, if you type:

```
?PRIMT "FOO  
I DON'T KNOW HOW TO PRIMT  
?ALSJKDFLKDFJKLFJ
```

the words **PRIMT**, **FOO**, and **ALSJKDFLKDFJKLFJ** will be created. However, if a word has no value, property list, or procedure definition, it will not be written to a file. So, if you are running out of space and have a lot of these words, you should use **RECYCLE**.

- It is possible to reserve space, but then it is your responsibility to keep track of that information.

Memory Allocation

When Logo starts up, it determines the amount of memory available on your IBM Personal Computer. This area is then partitioned: one part for nodespace, one part for Logo, and one part for DOS. See the memory map at the end of this chapter.

Note: Logo only uses up to 256K of memory. You can use any memory above 256K for your machine language subroutines without reserving it.

Reserving High End Memory Space

Logo provides the possibility of reserving space at the top of memory for your own purposes. This is necessary to use the .BLOAD and .CALL assembly languages primitives. You must start Logo from DOS, stating the amount of memory to be reserved. If you already are in Logo, give the .DOS command to enter DOS. Start Logo by typing

A>LOGO *n*

n stands for the amount of memory you want to reserve. If *n* is larger than the amount of memory that can be reserved by your system, the error message

NOT ENOUGH MEMORY

is printed when you enter Logo. For instance,

A>LOGO 10

reserves 10K at the top of the system's memory. If you have 128K of memory in your Personal Computer, this means Logo reserved memory starting at the 118thK or 128 - 10.

The address of this free memory is based at 7552. Use the following for calculating the address:

$base = (\text{amount of memory}) - n) * 1024 / 16$
 $maximum\ offset = (1024 * n) - 1$
(if n is less than or equal to 64)

$base$ is given by $118 * 1024 / 16$

- 118 is the amount of memory in your system minus the amount you reserved with LOGO n .
- 1024 is 1K of memory.

$offset$ will have a maximum value, in this case, of $(10 * 1024) - 1$

- 10 is the amount of memory reserved with LOGO n .
- 1024 is 1K of memory.

Other examples:

- Reserving memory with

A>LOGO 1

on a 128K system means that the address of this free memory is based at 8128.

$base$ is $127 * 1024 / 16$
maximum $offset$ is 1023

- Reserving memory with

A>LOGO 10

on a 192K system means that the address of this free memory is based at 11648.

base is $182 * 1024 / 16$

maximum *offset* is $(10 * 1024) - 1$

Values of Base and Offset

The *base* and *offset* are inputs to the assembly language primitives .BLOAD, .BSAVE, .CALL, .DEPOSIT, and .EXAMINE.

The *base* and *offset* do not accept values greater than 65535 or $2^{16} - 1$.

In general, to access any memory location, the address is divided into *base* and *offset*. For example, to access absolute memory location 120832:

base is $120832/16$ or 7552

offset is 0

Your inputs to the primitive .EXAMINE are

.EXAMINE 7552 0

The inputs for accessing address 120833 are

.EXAMINE 7552 1

You can access a byte of memory if you know the values of base and offset. To determine these values use the following equation:

$\text{Base} \times 16 + \text{offset} = \text{byte address}$

Many base and offset values can be used to access the same byte. For example, the byte address 8000 can be accessed by any of the following values:

```
.EXAMINE 0 8000
.EXAMINE 1 7984
.
.
.
.EXAMINE 499 16
.EXAMINE 500 0
```

Assembly Language Subroutines

You can use `.BLOAD` to load and `.CALL` to execute assembly language subroutines from `LOGO` if they have been converted to a format requiring no relocating (that is, segment fixup). The easiest way to get them into that form is to process your `.EXE` file (obtained from the linking process) with the `EXE2BIN.COM` to get a binary file.

1. Convert your linked program to a non-relocatable program file.

```
A>EXE2BIN PROG.EXE PROG.BIN
```

2. Start `Logo`, specifying the amount of memory you need for your program.

```
A>LOGO 10
```

3. Load your binary file. Because your file has no header, you must specify the *base* and *offset* where you want it to be loaded.

```
? .BLOAD "PROG.BIN 8000 0
```

Note that `.EXE2BIN` and `DEBUG` are *not* part of `Logo`, but are part of `DOS 2.00`. See your *IBM Personal Computer DOS 2.00 Reference* manual.

If you want to execute an assembly language program that needs to be relocated, you can do it by linking your program using the /H switch. You can then move your file in high memory using DEBUG. (High memory is where your reserved space is located.)

1. Link your program with the /H switch

```
A>LINK PROG.OBJ/H;
```

2. Load Logo under DEBUG

```
A>DEBUG LOGO.COM 1
```

3. Note the value of the SS, SP, CS, IP, and CX registers. Use the R command.

4. Use DEBUG to load your .EXE file into memory.

```
-N PROG.EXE  
-L
```

5. Display registers again using the R command. Note the hexadecimal values of CS, IP, and CX. These will be used later when you BSAVE your subroutine.

6. Reset the registers to their original values noted in Step 3.

7. Use the G command to start LOGO.

8. You can now BSAVE your subroutine. Convert the hexadecimal values of CS, IP, and CX from Step 5 to decimal. The CS register gives you the *base*, the IP gives the *offset*, and the CX the length of your subroutine.

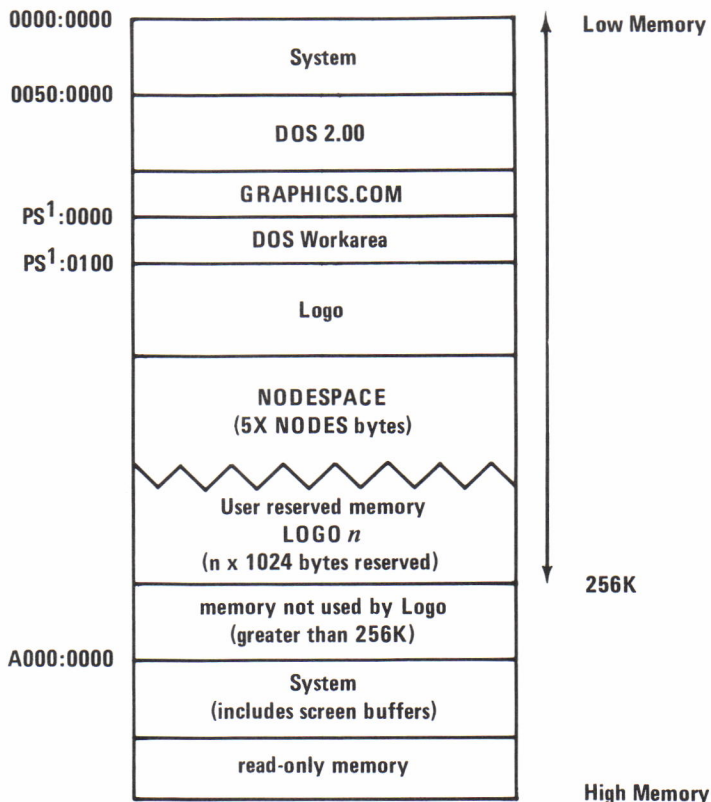
```
?.BSAVE "PROG.BIN 8084 0 128  
?.CALL 8084 0
```

To communicate with your subroutine, you should set aside locations in your code (with known *offsets*) and then use `.DEPOSIT` to pass parameters or `.EXAMINE` to receive results.

Logo provides a stack of 500 words. If you need more stack space, you can define your own. In any case, you must make sure that you leave the stack in the same state it was in when you received control.

Finally, you must remember that `.CALL` does a FAR call. (For more details, see the *IBM Personal Computer MACRO Assembler* manual.) Your procedure should then be coded as *proc far* so the return instruction can be coded accordingly.

Logo Memory Map



Notes:

1. PS¹ refers to DOS Program Segment.
2. The maximum NODESPACE is 163835 bytes.
3. The maximum number of NODES is 32767.
4. The maximum user reserved memory depends on the amount of memory available. When memory is reserved by the user, as with LOGO *n*, a minimum of 5K bytes of nodespace or 1K nodes is required.
5. If your computer has more than 256K bytes of memory, Logo will directly use only the first 256K bytes. However, you may use .EXAMINE and .DEPOSIT on memory locations after the first 256K bytes.

Appendix D. ASCII Character Codes

The following table lists all of the ASCII codes (in decimal) and their associated characters. These characters can be displayed using `PRINT CHAR n`, where *n* is the ASCII code. The ASCII characters are listed in the column headed “Character”. ASCII characters 0-33 can be displayed using Ctrl character key combinations. These are listed under the “Control Character” column.

Each of the ASCII characters can be entered from the keyboard by pressing and holding the Alt key, then pressing the digits for the ASCII code on the numeric keypad.

For certain keys or key combinations that cannot be represented in standard ASCII code, an extended code is returned by the primitives `READCHAR` or `READCHARS`. (See entry under each primitive for more details.) A null character (ASCII code 00) will be returned as the first character. If a 00 is received, then you should go back and examine the second character to determine the actual key pressed. Usually, but not always, this second code is the scan code of the primary key that was pressed. The ASCII codes (in decimal) for this second character and the associated key(s) are listed at the end of the appendix.

ASCII Value	Character	Control Character	ASCII Value	Character
000	(null)		033	!
001	☺	Ctrl A	034	"
002	☹	Ctrl B	035	#
003	♥	Ctrl C	036	\$
004	♦	Ctrl D	037	%
005	♣	Ctrl E	038	&
006	♠	Ctrl F	039	'
007	• (beep)	Ctrl G	040	(
008	■ (backspace)	Ctrl H	041)
009	○ (tab)	Ctrl I	042	*
010	⦿ (line feed)	Ctrl J	043	+
011	♂ (home)	Ctrl K	044	,
012	♀ (form feed)	Ctrl L	045	-
013	(carriage return)	Ctrl M	046	.
014	🎵	Ctrl N	047	/
015	☼	Ctrl O	048	0
016	▶	Ctrl P	049	1
017	◀	Ctrl Q	050	2
018	↕	Ctrl R	051	3
019	!!	Ctrl S	052	4
020	π	Ctrl T	053	5
021	§	Ctrl U	054	6
022	▬	Ctrl V	055	7
023	↑	Ctrl W	056	8
024	↑	Ctrl X	057	9
025	↓	Ctrl Y	058	:
026	→	Ctrl Z	059	;
027	←	Ctrl [060	<
028	↶ (cursor right)	Ctrl \	061	=
029	↷ (cursor left)	Ctrl]	062	>
030	▲ (cursor up)	Ctrl 6	063	?
031	▼ (cursor down)	Ctrl -		
032	(space)	Spacebar, Shift Spacebar, Ctrl Spacebar, Alt Spacebar		

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
064	@	096	'	128	Ç
065	A	097	a	129	ü
066	B	098	b	130	é
067	C	099	c	131	â
068	D	100	d	132	ä
069	E	101	e	133	à
070	F	102	f	134	å
071	G	103	g	135	ç
072	H	104	h	136	ê
073	I	105	i	137	ë
074	J	106	j	138	è
075	K	107	k	139	ï
076	L	108	l	140	î
077	M	109	m	141	ì
078	N	110	n	142	Ä
079	O	111	o	143	Å
080	P	112	p	144	É
081	Q	113	q	145	æ
082	R	114	r	146	Æ
083	S	115	s	147	ô
084	T	116	t	148	ö
085	U	117	u	149	ò
086	V	118	v	150	û
087	W	119	w	151	ù
088	X	120	x	152	ÿ
089	Y	121	y	153	Ö
090	Z	122	z	154	Ü
091	[123	{	155	•
092	\	124		156	£
093]	125	}	157	¥
094	^	126	~	158	Pt
095	_	127	□	159	f

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
160	á	192	Ł	224	α
161	í	193	ł	225	β
162	ó	194	ŧ	226	Γ
163	ú	195	ƒ	227	π
164	ñ	196	—	228	Σ
165	Ñ	197	+	229	σ
166	ä	198	ƒ	230	μ
167	ö	199	ƒ	231	τ
168	ÿ	200	ƒ	232	Φ
169	┘	201	ƒ	233	⊖
170	┐	202	ƒ	234	Ω
171	½	203	ƒ	235	δ
172	¼	204	ƒ	236	∞
173	ı	205	≡	237	∅
174	«	206	ƒ	238	€
175	»	207	ƒ	239	∩
176	▒	208	ƒ	240	≡
177	░	209	ƒ	241	±
178	▒	210	ƒ	242	≥
179		211	ƒ	243	≤
180	┘	212	ƒ	244	ƒ
181	┘	213	ƒ	245	J
182	┘	214	ƒ	246	÷
183	┘	215	ƒ	247	≈
184	┘	216	ƒ	248	°
185	┘	217	┘	249	•
186		218	┘	250	.
187	┘	219	■	251	√
188	┘	220	■	252	n
189	┘	221	■	253	²
190	┘	222	■	254	■
191	┘	223	■	255	(blank)

Extended Codes

Second Code	Meaning
3	(null character) NUL
15	(shift tab) ␣
16-25	Alt Q, W, E, R, T, Y, U, I, O, P
30-38	Alt A, S, D, F, G, H, J, K, L
44-50	Alt Z, X, C, V, B, N, M
59-62	function keys F1 through F4
64-68	function keys F6 through F10
71	Home
72	Cursor Up
73	Pg Up
75	Cursor Left
77	Cursor Right
79	End
80	Cursor Down
81	Pg Dn
82	Ins
83	Del
84-93	F11-F20 (Shift F1 through F10)
94-103	F21-F30 (Ctrl F1 through F10)
104-113	F31-F40 (Alt F1 through F10)
114	Ctrl PrtSc
115	Ctrl Cursor Left (Previous Word)
116	Ctrl Cursor Right (Next Word)
117	Ctrl End
118	Ctrl Pg Dn
119	Ctrl Home
120-131	Alt 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =
132	Ctrl Pg Up

Appendix E. Startup File Feature

A special feature of Logo allows you to automatically load a file into your workspace at the same time that you start Logo. This file must be called **STARTUP.LF**. Right now, you do not have the **STARTUP.LF** file on your Logo Language Diskette. **STARTUP.LF** must be on the diskette in the default drive as Logo will search only this drive for **STARTUP.LF**.

Here are some examples of how you can create the file.

- There is a file called **TOOLS.LF** on your Logo Language Diskette. This file contains the procedures listed in Appendix B. If you would like to automatically load this file into your workspace each time you start Logo, do the following:

LOAD "TOOLS

to load the **TOOLS.LF** file into your workspace.

POTS

to check if all of the procedure titles are listed.

SAVE "STARTUP

to create a file called **STARTUP.LF** containing the procedures from the **TOOLS** file.

Now try it out by restarting Logo. When you see “**WELCOME TO LOGO**” on the screen, print out the procedure titles. All of the **TOOLS** procedures should be in your workspace.

- You may want to automatically load only some of the procedures contained in the TOOLS file. If so, do the following:

```
LOAD "TOOLS
```

to load the TOOLS.LF file.

```
PACKAGE "AIDS [CIRCLER CIRCLEL ARCR AR→  
CL]
```

to group the procedures you want to save into the STARTUP file. We are packaging only four procedures.

```
SAVE "STARTUP "AIDS
```

to save the AIDS package into the STARTUP.LF file.

Now restart Logo to check if the correct procedures are automatically loaded.

- You may want to automatically load the same procedures as in the previous example, but you don't want the user to mix them with his or her procedures. For instance, you want to create a special Logo environment that has circles and arcs with radius as part of the primitives. In this case,

```
LOAD "TOOLS
```

```
PACKAGE "AIDS [CIRCLER CIRCLEL ARCR AR→  
CL]
```

```
BURY "AIDS
```

to bury the procedures in the AIDS package.

```
SAVE "STARTUP "AIDS
```

Now restart Logo. When you try POTS after “WELCOME TO LOGO” appears on the screen, nothing should be printed. Try

```
POTS "AIDS
```

The procedures you saved in STARTUP are printed.

We have provided only three examples of using the STARTUP file feature. Note that you can save *any* procedures you want in this file, and Logo will load them automatically.

Startup Variable

There is also a **STARTUP** system variable (not to be confused with the **STARTUP** file).

If the **STARTUP** variable contains a list of procedures or commands, then Logo runs the list immediately after loading the file containing the variable. For example:

```
?MAKE "STARTUP [PRINT [HERE IT IS]]  
?SAVE "ANYNAME  
0 PROCEDURES SAVED  
?LOAD "ANYNAME  
HERE IT IS
```

If your **STARTUP** file contains a **STARTUP** variable which is a list, it will be executed on starting Logo. The **STARTUP** variable is especially useful for automatically executing procedures.

Appendix F. Your Logo Language Diskette and DOS

Replacing DOS on the Logo Diskette

Your Logo Language Diskette comes with DOS 2.00 already on the diskette. If you would like to use Logo with a newer version of DOS, you can prepare a self-starting Logo Language Diskette with the new version of DOS in the following manner:

1. Insert the DOS diskette in drive A and close the door.
2. Switch on the System Unit and the monitor. If your machine is already on, hold down the following three keys at the same time: Ctrl, Alt, and Del.
3. Type in the date and the time to get the DOS prompt `A>`.
4. Type:

`FORMAT B:/S`

Insert a blank diskette when prompted. This formats the diskette and transfers the new version of DOS.

Note: When you format a diskette, all existing information is erased. To avoid losing important data use a blank diskette or one that contains obsolete information.

5. Use the DOS COPY command to copy the external DOS command files. You should copy GRAPHICS.COM, DISKCOPY.COM, and FORMAT.COM, along with any other DOS external command files you want on your new Logo Language Diskette.

- If you have a two-drive system, insert the DOS diskette in drive A and the new, formatted diskette in drive B. Type

`COPY A:GRAPHICS.COM B:`

to copy the DOS GRAPHICS command file from the DOS diskette to the new, formatted diskette. Follow the same procedure for copying DISKCOPY.COM and FORMAT.COM.

- If you have a one-drive system, insert the DOS diskette in drive A and type

`COPY A:GRAPHICS.COM B:`

to copy this command file from the DOS diskette to the new, formatted diskette. Follow the DOS prompts to insert the DOS diskette and the new, formatted diskette in the drive.

6. Use the DOS COPY command to copy the LOGO command file.
 - If you have a two-drive system, put the old Logo Language Diskette (make sure that it has a write protect tab) in drive A and the new, formatted diskette in drive B. Type

`COPY A:LOGO.COM B:`

to copy the Logo command file from the old Logo Language Diskette to the new, formatted diskette, which becomes your new Logo Language Diskette with the upgraded version of DOS.

- If you have a one-drive system, put the old Logo Language Diskette (make sure that it has a write protect tab) in drive A and type

`COPY A:LOGO.COM B:`

to copy the Logo command file from the old Logo Language Diskette to the new, formatted diskette. Follow the DOS prompts to insert the old Logo Language Diskette and the new, formatted diskette in the drive.

How to Start Logo When DOS is Loaded

If you have already started your computer and it displays the IBM Personal Computer DOS prompt **A>**, start Logo by inserting the Logo Language Diskette in drive A and typing:

```
AUTOEXEC
```

Editing the AUTOEXEC

Any program listed in the AUTOEXEC file is automatically executed when the Logo Language Diskette is started.

The AUTOEXEC file can be edited with Logo by typing

```
EDITFILE "AUTOEXEC.BAT
```

and using the Logo Editor to change the file.

Graphics Program

One of the programs on the Logo Language Diskette is GRAPHICS, a DOS command which supports the IBM Personal Computer Graphics Printer. Before Logo begins, it is loaded into user memory by the AUTOEXEC.BAT file.

Without the GRAPHICS program loaded into memory, you would be unable to use the SAVEPIC command or the Shift-PrtSc keys to print graphics screens on the graphics printer. You would still be able to use SAVEPIC and Shift-PrtSc to print the textscreen.

If you do NOT have the IBM Personal Computer Graphics Printer, the GRAPHICS program is not required and should be removed from the AUTOEXEC.BAT file. You then will be able to use the SAVEPIC command and Shift-PrtSc keys to print the textscreen and the text portion of the graphics screen on the printer.

Mode Command

The DOS MODE command can be used to direct output from the IBM Personal Computer or IBM Personal Computer XT to a serial printer. If the MODE command is included in the AUTOEXEC, Logo will use the MODE parameters, and you will not need to use .SETCOM to set up the serial printer information. Refer to the *IBM Personal Computer DOS Reference* for more information on the MODE parameters and syntax.

Glossary

The glossary contains definitions of important terms used in this book. Some terms defined in the text of the book may not appear.

adapter: A mechanism for attaching parts.

address: The location of a register, a particular part of memory, or some other data source or destination. Or, to refer to a device or a data item by its address.

allocate: To assign a resource, such as a disk file or a part of memory, to a specific task.

alphabetic character: A letter of the alphabet.

ASCII: American National Standard Code for Information Interchange. The standard code used for exchanging information about data processing systems and associated equipment. An ASCII file is a text file where the characters are represented in ASCII codes.

background: The part of the display screen surrounding a character or a graphic design.

backup: A system, device, file, or facility that can be used in case of a malfunction or loss of data.

baud: A unit of signaling speed equal to the number of discrete conditions or signal events per second.

binary: Something that has two possible values or states. Also, refers to the Base 2 numbering system.

bit: A binary digit.

blinking: A regular change in the intensity of a character on the screen.

boot: Load the language into the computer's memory when you start up Logo.

buffer: An area for storage of data which is used when transferring data from one device to another. Usually refers to an area reserved for an input/output operation, into which data is read or from which data is written.

bug: An error in a program.

byte: Eight bits.

call: To bring a computer program, a procedure, or a subprocedure into effect.

carriage return character (CR): A character that causes the print or display position to move to the first position on the same line.

channel: A path along which signals can be sent; for example, a data channel or an output channel.

character: A letter, digit, or other symbol that is used as part of the organization, control, or representation of data.

command: A procedure that has no output.

communication: The transmission and reception of data.

constant: A fixed value or data item.

coordinates: Numbers which identify a location on the display screen.

cursor: A movable marker that is used to indicate a position on the display screen.

debug: To find and eliminate mistakes in a program.

default: A value or option that is assumed when none is specified.

device: Anything attached to the computer, such as a printer, display, or diskette drive.

directory: A table, for a disk, that contains the names of files (identifiers) on the disk, along with information that tells DOS where to find the files on the disk.

disabled: A state that prevents the occurrence of certain types of actions, such as the normal functioning of the Caps Lock key once Logo is started up.

DOS: Disk Operating System. In this book, refers only to the IBM Personal Computer Disk Operating System.

echo: To reflect received data to the sender. For example, keys pressed on the keyboard are usually echoed as characters displayed on the screen.

edit: To enter, modify, or delete data.

element: A member of a set; in particular, an item in a series.

empty: Having no meaning or nothing in it. In particular, a word or a list with no elements in it.

erase: To remove information permanently from either the workspace or a file.

execute: To perform an instruction or a computer program.

file: An organized collection of information that can be permanently stored for specific purposes.

format: The particular arrangement or layout of data on a data medium, such as the screen or a disk.

function: The specific purpose of an entity, or its characteristic action.

function key: One of the ten keys labeled **F1** through **F10** on the left side of the keyboard.

garbage collection: Cleaning the computer's memory to make more space available for storage.

global variable: A variable that is always in the workspace, such as variables you create with the **MAKE** primitive.

grammar: The rules by which Logo instructions are written.

graphic: A symbol produced by a process such as handwriting, printing, or drawing.

hard copy: A printed copy of machine output in a visually readable form.

hierarchy: A structure having several levels, arranged in a tree-like form. "Hierarchy of operations" refers to the relative priority assigned to arithmetic or logical operations that must be performed.

input: The information that a Logo primitive or procedure needs in order to begin execution.

instruction: In a programming language, any meaningful expression that specifies one command and its inputs.

integer: One of the numbers 0, ± 1 , ± 2 , ± 3 , ...

interactive: A program that creates a dialogue between the computer and the user.

interface: A shared boundary.

interrupt: To stop a process in such a way that it can be resumed.

joystick: A lever that can pivot in all directions and is used as a locator device.

K: When referring to storage capacity, two to the tenth power or 1024 in decimal notation.

line feed (LF): A character that causes the print or display position to move to the corresponding position on the next line.

list: In Logo, a sequence of words or lists that begins and ends with brackets.

literal: An explicit representation of a value, especially the value of a word or a list; a constant.

local variable: A variable that exists only when a procedure is being executed.

location: Any place in which data may be stored.

loop: A set of instructions that may be executed repeatedly while a certain condition is true.

name: A word that is being used as a container for something in the workspace.

node: A place in the memory of the computer that is a signpost to other parts of the memory.

notation: A set of symbols, and the rules for their use, for the representation of data.

operation: A procedure that has some kind of output.

output: The information that a Logo primitive or procedure attempts to give to another primitive or procedure.

package: A collection of variables and procedures.

picture element (PIXEL): A graphics “point.” Also, the bits that contain the information for that point.

port: An access point for data entry or exit.

position: In a word or a list, each location that may be occupied by an element and that may be identified by a number. For the graphics screen, a location specifying the coordinates of the position of the turtle in the form of a list [x y].

precision: A measure of the ability to distinguish between nearly equal values.

predicate: A procedure that outputs either TRUE or FALSE.

primitive: A procedure that is built into Logo.

procedure: A sequence of instructions, that has a name, which can be permanently stored.

program: A set of procedures that work together.

prompt: A question the computer asks when it wants you to supply information.

range: The set of values a quantity or function may take.

read: To input data into a device so you can have access to it. For example, see the LOAD primitive.

real number: Any positive or negative decimal number.

record: A collection of related information treated as a unit. For example, in stock control each invoice may be one record.

recursive: A procedure that calls itself as a subprocedure.

resolution: In computer graphics, a measure of the sharpness of an image, expressed as the number of lines per unit of length discernible in that area.

routine: Part of a program, or a sequence of instructions called by a program, that may have some general or frequent use.

scan: To examine sequentially, part by part.

scroll: To move all or part of the display image vertically or horizontally so that new data appears at one edge as old data disappears at the opposite edge.

segment: A particular 64K-byte area of memory.

stack: A method of temporarily storing data so that the last item stored is the first item to be processed.

sublist: A list that is inside another list.

subprocedure: A procedure whose name occurs in the definition of another procedure.

superprocedure: A procedure that contains one or more subprocedures.

storage: A device, or part of a device, that can retain data.

string: A sequence of characters.

syntax: The rules governing the structure of a language.

table: An arrangement of data in rows and columns.

terminal: A device, usually equipped with a keyboard and display screen, capable of sending and receiving information.

top level: The mode in which commands can be executed directly without being embedded in a program.

truncate: To remove the ending elements from a word.

turtle: The shape on the screen that represents the pen that Logo uses to draw lines.

typematic key: A key that repeats as long as you hold it down.

update: Usually to modify a master file with current information.

variable: A word that can assume any of a given set of values.

workspace: The part of the computer's memory that lasts only as long as the computer is switched on. It is used to hold variables and procedures.

wrap: The technique for displaying graphics that has coordinates lying outside of the display area.

write: To record data in a data medium.

Index

Special Characters

- * asterisk 2-17, 5-16, 7-18
- \backslash 2-17
- [] brackets 1-8, 2-16
- : colon 2-6, 2-10, 2-15, 7-171
- continuation arrow 2-14, 3-4
- / division sign 5-16, 7-19
- = equal sign 2-15, 5-16, 7-22
- > greater than 2-17, 5-16, 7-21
- < less than 2-17, 5-16, 7-20
- minus sign 2-17, 5-16, 7-15
- () parentheses 1-8, 2-17, 7-4
- . period 4-4
- + plus sign 2-17, 5-16, 7-14
- “ quotes 2-6, 2-18, 5-3
- space 2-17

See also Special Keys

A

- abbreviations 7-5
- ABS 7-16, 7-89, 7-190, B-1
- absolute value 7-89
- accessing
 - files 7-124
 - memory C-7
 - variables 7-171
- active
 - caps 7-58
 - partition 1-24

- adapter
 - Asynchronous Communications 1-4, 4-5, 7-35
 - communications 7-35
 - Games Control 7-57
 - parallel printer 1-3
- ADD 7-316
- adding
 - data to a file 4-14
 - procedures to a package 4-9, 7-191
 - variables to a package 7-205, 7-223
- ADDIT B-11
- addition 7-14, 7-316
- ADDLINES B-10
- address 7-26, C-6
- ADD.QUIZ 7-147
- ADD.RHYME 7-134
- AGE 7-245
- AIDS package E-2
- ALLOPEN 7-39
- Alt key D-1
- AND 7-40
- angles 5-19, 7-42, 7-203
- ANNOUNCE 7-328
- ANYBASE.TO.DEC B-2
- ARCCOS 7-45
- ARCL B-3
- ARCR B-2
- ARCSIN 7-45
- ARCTAN 5-19, 7-42
- arithmetic operations 5-3, 5-16, 6-6
- arrow, continuation 2-14, 3-5
- ASCII 7-47, 7-63, D-1
- ASKINFO 4-14
- aspect ratio 7-34, 7-37

assembly language 4-19,
7-26, C-7

assigning values to
variables 7-171

Asynchronous

Communications

Adapter 1-4, 4-5, 7-35

AUTOEXEC file F-4

AUX device 4-5

B

BACK 7-49

BACKGROUND (BG) 7-51,
7-271

Backslash key 2-17, 7-229

Backspace key 1-9, 3-7,
7-9, D-2

backup diskette 1-13

base 7-23, 7-28, C-6

baud rate (Bdrt) 7-35

BETWEEN 7-20

BIGWELCOME 2-5

binary

file 4-7, 4-19

load 7-23

save 7-25

BLACK.BLUE B-12

BLACK.BROWN B-12

BLACK.CYAN B-12

BLACK.GREEN B-12

BLACK.MAGENTA B-12

BLACK.RED B-12

BLACK.WHITE B-12

.BLOAD 4-20, 7-23,
C-5, C-7

BOUNCE 7-279

BOUNDS 7-303

bounds of screen 7-339, 7-344

Bracket keys 1-8, 2-6, 2-17

break 1-7, 3-9, 3-13

.BSAVE 7-25

buffer

delete 3-5, 3-7

edit 3-4, 7-97, 7-98, 7-99

video 7-166

bugs (debugging) 4-12, 7-116,
B-8

BURY 7-52, E-3

BUTFIRST (BF) 7-54

BUTLAST (BL) 7-56

BUTTONP 7-57

BYE 7-39

BYTEPOS B-6

bytevalue 7-28

C

Caesar Cipher 7-48

CALCULATOR 7-260

.CALL 7-26, C-5, C-7

call a procedure 2-4

CANCEL 7-84, 7-88

CAPITAL 7-180

CAPITAL.QUIZ 7-146

CAPS 7-58, 7-272

Caps Lock key 1-7,
7-58, 7-272

CATCH 7-60, 7-117, 7-326

CHANGEGB 7-271, 7-337

CHAR 7-63

CHAR 32 7-334, B-8

characters

ASCII codes D-1

blinking 7-293, 7-320

converting 7-48, 7-63

detecting 7-151

dribble file 4-12

keys 1-6

quoting delimiters 2-17

reading 7-235, 7-238

underlined 7-293, 7-321

CHECK 7-58, 7-343
 CHECKLIST 7-160
 CHECKPOS 7-347
 CHECKREAD 7-241
 CHECKWRITE 7-348
 CIRCLE 7-38, 7-142
 CIRCLEL B-2
 CIRCLER B-2
 CLEAN 7-64
 CLEARSCREEN (CS) 7-65
 CLEARTEXT (CT) 7-66
 CLOSE 4-15, 7-69
 CLOSEALL 7-39, 7-71
 CO (COntinue) 7-72,
 7-117, 7-195
 colon 2-6, 2-9, 2-15,
 7-83, 7-171
 color
 background 7-51, 7-271
 pencolor 7-198, 7-281
 text 7-292, 7-320
 colorlist 7-6, 7-292
 colnumber 7-6
 columns of numbers 7-103,
 7-129
 columns on the screen 7-273,
 7-297
 COM Extension 1-24, C-8
 COM1 device 4-5, 4-20
 COM2 device 4-5, 4-20
 commands
 DOS F-4
 editor 3-8, 7-97
 Logo 2-8, 5-13, F-2
 comments (TO ;) B-1
 communications 1-4,
 4-5, 7-35
 CON (CONsole device) 4-15,
 7-298

.CONTENTS 7-27, C-4
 continuation character 2-4
 control characters 4-19
 Control key See Ctrl Key
 CONVERT B-3
 coordinates, x and y 7-42,
 7-93, 7-349, 7-350, 7-351
 COPY B-12
 COPYDEF 7-75, 7-83, B-12
 copying
 Logo diskette 1-13
 Logo line 3-13
 memory to disk 7-25
 COS 7-78
 cosecant 7-79
 COSINE 7-350
 cotan 7-79
 COUNT 7-81
 COUNTD 7-138
 COUNTDOWN 7-137
 COUNTUP 7-14, 7-72
 creating a data file 4-14
 C.TO.N B-3
 Ctrl key
 Alt-Del 1-8
 ASCII D-1
 Break 1-7, 3-9, 7-10, 7-98,
 7-195
 Cursor 3-7, 7-9
 Numlock 7-10
 PgDn 3-6, 7-9
 CUBE 7-232
 CURSOR 7-82
 Cursor key 1-8, 3-4, 3-5,
 7-9, 1-13
 CURVE 7-93
 CYAN B-7

D

data 7-24, 7-31
databits 7-35
date 1-11, 7-91
debugging hints 4-12, 7-86,
7-116, B-8
DECAY 7-120
DECIDE 7-144
decimal point 5-5, 7-22, 7-39,
7-129, B-3
DECIMALP 7-41
DEC.TO.ANYBASE B-3
DECTOHEX B-3
default
disk drive 4-3, 7-90
extension 4-4
number of screens 7-33
precision 5-10
screen width 7-296
DEFINE 7-83
DEFINEDP 7-87
defining procedures 2-3,
7-324
degrees 5-19, 7-6, 7-42,
7-140, 7-207
Del key 1-9, 3-7, 7-10
delete buffer 3-5, 3-7
deleting
characters 3-4
end of line 3-12
files 7-111
delimiters 2-17
DEPO 7-29
.DEPOSIT 7-28, C-7
DESIGN 7-283
destroying workspace 7-26
device 7-6
device names 4-5
devices 4-5, 4-14
DIAMONDBACK 7-310

DICE 7-253
DICE6 7-253
dictionary 7-170
DIFFERENCE 7-15, 7-89
DIR (DIRectory) 4-8, 7-90
DISK 7-92
disk files (see also files)
editing 7-97
erasing 7-111
existence 7-123
names 4-3
printing 7-96, 7-213
diskette
care of 1-5
copying 1-13
creating file diskette 1-19
DOS F-1
source 1-15
target 1-16
DISPBYTE B-6
DISTANCE 7-312
Division key 7-19, 7-233
DIVISORP 7-248, B-4
DOIT 7-62
DOIT1 7-62
.DOS 7-30
DOS 2.00 F-1
DOT 7-93
DRIBBLE 4-12, 4-21, 7-94
DRIVE 7-57, 7-236,
7-326, B-4
DUMBTURTLE 7-291
dummy device 4-5
DUMP 4-21, 7-96, 7-213
duration 7-6

E

ECHO 7-74
EDIT (ED) 3-3, 7-97
EDITFILE 7-99

editing 3-4
 EDITOR 7-100
 EDNS (EDit NameS) 7-97
 EFORM 7-103
 ELAPSED TIME 7-32
 elements
 of a list 2-18, 7-81
 of a word 7-81
 ELLIPSE 7-37
 empty list 2-19, 7-104
 empty word 2-19, 7-104
 EMPTY 7-104
 END 2-3
 End key 7-10
 end of file 7-240, 7-345
 ending Logo 7-27
 enter key 1-6, 2-14, 3-5, 7-9
 equal key 2-17, 7-22
 EQUALP 7-22, 7-106
 ERALL 4-10, 7-108, 7-222
 ERASE (ER) 7-110
 erase to end of line 3-12
 ERASEFILE 4-10, 7-111
 ERN (ERase Name) 7-113
 ERNS (ERase NameS) 2-13,
 4-8, 7-114
 ERPS (ER ProcS) 7-115
 ERRACT 7-52, 7-116, 7-207
 ERROR 7-60, 7-117
 error messages A-1
 trapping 7-60, 7-117
 Esc key 3-8, 7-10, 7-97,
 7-100, D-2
 ESTIMATE 7-286
 EVENP 7-248
 .EXAMINE 7-31, C-7
 EXE2BIN C-8
 exit logo 7-30
 EXP 7-119, 7-161, 7-220
 exponent 5-5

exponential function 7-119,
 7-220
 ext 7-6
 .ext 4-4
 extended code 7-235, D-5
 extensions 4-4, 7-90
 .BIN C-8
 .EXE C-8
 .ext 4-4
 .LF 7-265
 NUL 4-5
 null 7-235, D-1, D-2
 .PIC 7-267

F

F1 (TS) 7-10, 7-322
 F2 (MS) 7-10, 7-66, 7-176
 F3 (repeat last line) 3-7, 7-10
 F4 (FS) 7-10, 7-66, 7-135
 F5 7-10, 7-73, 7-195
 FACTORIAL 7-18
 FALSE 7-142
 FASTCIRCLE 7-306
 FEATURE 7-76
 FENCE 7-122
 file 4-3
 file header 7-23, 7-25
 file name 4-3
 file pos 7-6
 FILELEN 7-123
 FILEP 7-124
 FILERL 7-290
 files (see also disk files) 4-6
 binary 4-19
 data 4-13
 directory 4-8
 disk 4-6
 dribble 4-12
 program 4-8
 screen 4-11

filespec (file specification) 4-3
 FILL 7-125
 FILLIN 7-123
 FILLSQ 7-125
 FINDINFO 4-17
 FINDTEL 4-17, 4-19
 FIRST 7-127
 Fixed disk 1-24
 FLAG 7-88
 FLATTER 7-270
 FLAVORCHART 7-82
 FLIP 2-9, 7-22
 floating point format 5-5
 FOREVER 7-263, 7-315,
 B-4
 FORM 7-129
 FORMAT a diskette 1-16,
 7-30, F-1
 format of numbers 5-5
 FORWARD (FD) 7-132
 FPUT 7-134, 7-159
 free nodes 7-181
 frequency 7-7
 FROM.HOME 7-259, 7-312
 FULLSCREEN (FS) 7-135
 function keys 1-10
 functions (see also logic
 and predicates)
 ABS B-1
 ARCCOS 7-45
 ARCSIN 7-45
 ARCTAN 7-42
 COS 7-79
 EXP 7-119
 HEXTODEC B-3
 INT 7-148
 LN 7-161
 LOG2 7-162
 POWER 7-220
 PRODUCT 7-231

QUOTIENT 7-233
 RANDOM 7-234
 REMAINDER 7-247
 ROUND 7-258
 secant 7-79
 SIN 7-308
 SINE 7-352
 SQRT 7-312
 SUM 7-316
 TAN 7-79

G

Games Control Adapter 7-57,
 7-193
 garbage collection 7-246, C-1
 GETEC 7-237
 GET.USER 7-242
 global variables 2-12
 GO 7-138, 7-152
 GOODVEE 7-217
 GPROP (Get
 PROProperty) 7-139
 grammar, Logo 2-3
 graphics
 adapters 1-3
 printer 1-12, 4-21
 setting to a separate
 screen 7-36
 GRAPHICS 1-12, 7-33, F-4
 GRAPHICS.COM 1-3,
 7-30, F-2
 graphics screen
 erasing 7-64, 7-65
 saving 4-9, 4-20, 7-267
 greater than 7-21
 GREEN B-7
 GREET 7-179, 7-207
 GUESSNUM 3-9, 7-89

H

HEADING 5-19, 7-140
hexadecimal numbers B-3
HEXTODEC B-3
HIDETURTLE (HT) 7-142
HOME 7-143
Home key 7-10

I

IF 7-144
IFFALSE (IFF) 7-146
IFTRUE (IFT) 7-147
IGNORE 7-77
INC 7-313, 7-325
infilespec 7-7
infix operation 7-14
INP 7-107, 7-190
inputs 2-5, 7-5
 to primitives 7-8
 to procedures 2-5
 to TO 7-328
Ins key 3-7, 7-11
inserting characters 3-7
instruction list 2-7, 7-7
INT (INTEger) 7-148
integer 7-148
INTP 7-148
invisible turtle 7-142
isosceles triangle 7-143
ITEM 7-150

J

joysticks 7-57, 7-193

K

keyboard 1-6, 4-5
KEYP 7-151
keys
 Backspace 1-9, 3-7,
 7-9, D-2
 Cursor 1-8, 1-13, 3-4,
 3-5, 7-9
 Delete 1-9, 3-7, 7-10
 Enter 1-6, 2-14, 3-5, 7-9
 Function 1-10
 Insert 3-7, 7-11
 Num Lock 1-9, 7-11
 Page down 3-8
 Page up 3-8
 Semi-colon B-1
 Space bar 1-9

L

LABEL 7-137, 7-157
LABELART 7-314
labelling pictures 7-314
LABELPIC 7-276
LAST 7-153
LATIN 7-341
LEARN 7-84, 7-87
LEFT (LT) 7-155
LEG 7-85
length
 edit buffer 7-99
 file 7-123
LENGTH 7-208
.LF (Logo File) 4-4,
 7-164, 7-265
LF (Line Feed) D-2

LIST 7-157
LISTEN 7-326, B-4
LISTFILE 7-243
LIST.OF.DOTS 7-310
LISTP 7-160
lists 2-7, 2-14
literal word 2-6, 2-10
LN 7-161
load (see .BLOAD,
LOADPIC, LOAD)
LOAD 4-10, 7-164
LOADPIC 4-12, 7-166
LOCAL 7-167
local variables 2-12, 7-167
logarithm (log) 7-162
logarithms (natural) 7-119,
7-161
LOG2 7-162
logic
 AND 7-40
 NOT 7-183
 OR 7-187
Logo grammar 2-3
Logo lines 2-14
LOGO.COM 1-24
LOWERCASE 7-63
lowercase characters 1-7,
7-59
LPT1 device 4-5
LPT2 device 4-5
LPT3 device 4-5
LPUT 7-159, 7-169

M

MAKE 2-11, 7-171
MAP 7-262, B-5
MARK.TWAIN 7-189

mathematical identities 7-46,
7-79, 7-121, 7-163
MEMBERP 7-174
memory
 addresses C-6
 high end C-5
 map C-11
 nodes C-1
 random access 1-3
 reclaiming 7-246, C-1
MESSAGE 7-230
META 7-77
Minus-sign key 2-17, 5-5,
5-18, 7-15
MIXEDSCREEN
 (MS) 7-176, 7-275
MODE command F-5
MODIFY 4-18
MODINFO 4-19
monitor
 selecting 7-36
 types 1-3
 width 7-296
MOUNTAINS 7-188
MOVE 7-335
MOVECURSOR 7-274
multiplication 7-18, 7-232

N

NAME 7-178
NAMEIT B-11
namelist 7-7
NAMEP 7-180, 7-325
naming files 4-3
NEAR 7-17
NEWENTRY 7-170
newname 7-7
nodes C-1

NODES 7-182, 7-246, C-2
NODRIBBLE 4-13, 7-182
NOT 7-183
notes (musical) 7-330
N.TO.C B-3
NUL device 4-5
null extension 7-111
Num Lock key 1-9, 7-10
NUMBER 7-285, 7-302
numeric keypad 1-8, 1-9
NUMBERP 7-185
numbers
 floating point 5-5
 input of 5-6
 negative 5-8
 output of 5-9
 scientific 5-9
NUMLISTP 7-185

O

object 7-7
offset 7-23, 7-26, C-6
OFFSET B-3
ONEUP 7-56
OPEN 7-186
operations 2-8
OR 7-187
organizing procedures in
 files 4-6
outfilespec 7-7
OUTPUT (OP) 7-189
OUTSIDE 7-21

P

PACKAGE 4-9, 7-52, 7-191
packagelist 7-7
PADDLE 7-193

paddlenumber 7-7
Page Down key (see PgDn key)
Page Up key (see PgUp key)
PALETTE (PAL) 7-194
Parentheses key 1-8
parity 7-35
parsing a Logo line 2-14, 7-77
Part Screen key (see PrtSc key)
PAUSE 7-195
PDRAW 7-193
PEEK (See .EXAMINE)
PEL B-6
PEN 7-197
PENCOLOR (PC) 7-198
PENDOWN (PD) 7-199
PENERASE (PE) 7-200
penlist 7-8
PENREVERSE (PX) 7-201
PENUP (PU) 7-202
PgDn key 3-8, 7-11
PgUp key 3-8, 7-11
PI 7-203
.PIC extension 7-267
PICK 6-5, 6-12
PIECE 7-283
PIG 7-341
PKGALL 7-205
PLACET 7-302
PLIST 7-207
Plus-sign key 2-17, 7-14
PO (Print Out) 7-208
POALL 7-209
POFILE 7-212
POKE (See .DEPOSIT)
POLY 7-141, 7-155, B-8
POLYCEN 7-309
POLYGON 7-19
PONS (PO NameS) 7-214
POPS (PO ProcS) 7-215
POS (POSition) 7-217
position 7-8

- POSITIVE 7-145
- POTS (PO TitleS) 7-219
- POWER 7-220
- PPROP (Put
PROPERTY) 7-222
- PPS (Print PropS) 7-224
- PRDOLLARS 7-131
- precedence of operations 5-16
- PRECISION 5-13, 7-226
- pred 7-8
- predicates
 - BETWEEN 7-20
 - BUTTONP 7-57
 - CAPS 7-58
 - DEFINEDP 7-87
 - EMPTYTYP 7-104
 - EQUALP 7-106
 - FILEP 7-124
 - INP 7-107
 - INTP 7-149
 - KEYP 7-151
 - LISTP 7-160
 - MEMBERP 7-174
 - OUTSIDE 7-21
 - PRIMITIVEP 7-228
 - REALWORDP 7-184
 - SHOWNP 7-306
 - VOWELP 7-175
 - WORD? 7-183
 - WORDP 7-342
- PRIMITIVEP 7-228
- primitives 2-3
- PRINT (PR) 7-229
- PRINTBACK 7-154
- PRINTDOWN 7-128
- printer
 - graphics 4-21
 - parallel 4-20
 - serial 4-20, 7-35, 7-96

- printing
 - files 4-20
 - to parallel printer 4-20
 - to printer 4-22, 7-230
 - to serial printer 4-20
 - screen 4-19, 7-248
- PRINTMESSAGES 3-5
- PRINTVARS 7-324
- PRN device 4-5
- PROBGEN 6-8, 6-11
- procedure call 2-4
- procedures 2-4, 2-5
- PROCPKG
 - property 7-191, 7-205
- PRODUCT 7-231
- project, developing 6-3
- PROMPT 7-334, B-8
- prompt symbol 1-12
- prop 7-8
- property lists 7-52, 7-191,
7-207, 7-249
- PrtSc key 4-21, 7-11

Q

- quotes 2-6
- QUOTIENT 7-233

R

- radians 7-203
- RANDOM 7-234
- RANDOMBOX 7-285,
7-333
- RANPICK 7-81, 7-150
- READCHAR (RC) 7-235
- READCHARS (RCS) 7-238

READEOFP (READ End
 of File Position) 7-240
 READER 7-241
 READFILE 7-186, 7-240
 READLINES 7-84, 7-87
 READLIST 7-242
 READNUM 7-61
 READPOS 7-243
 READTIME 7-32
 READWORD (RW) 7-244
 REALWORDP 7-184
 recursive
 procedures 6-7, 7-138
 RECYCLE 7-242
 RED B-7
 redefining a primitive 7-83
 REDEFP 7-52, 7-75, 7-83
 REMAINDER 7-247
 REMPROP 7-249
 REPARSE 7-250
 REPEAT 7-251
 replacing turtle shape 7-291
 REPORT 7-337
 REPRINT 7-230
 RERANDOM 7-253
 RESTORE 7-283
 retrieving information 4-17
 returning control 7-61, 7-315
 REVPRINT 7-105
 RIGHT (RT) 7-256
 ROUND 7-258
 rounding of numbers 5-10
 roundoff errors 7-46
 RTTRI 7-143
 RUN 7-260
 RUNSTORE B-11

S

SAFE.SQUARE 7-264
 SAFESQUARE 7-118
 SAVE 4-4, 4-20, 7-265
 SAVEINFO 4-15
 SAVEPIC 4-11, 4-21, 7-267
 scientific notation 5-7, 7-103
 .SCREEN 7-33
 screen
 boundaries 7-122,
 7-339, 7-344
 hardcopy 4-21, 7-248
 modes 7-33, 7-35,
 7-135, 7-176, 7-255, 7-322
 saving contents 4-11
 4-20, 7-267
 shifting 7-275, 7-313
 two displays 7-33, 7-36
 width 7-296, 7-338
 SCRIBBLE 4-22
 scrolling the screen 3-8
 .SCRUNCH 7-34, B-5
 secant 7-79
 SECRETCODE 7-48
 SECRETCODELET 7-48
 Semi-colon key B-1
 SENTENCE (SE) 7-159,
 7-269
 serial printer 4-20, 7-35, 7-96
 SETBG 7-271
 SETCAPS 7-272
 .SETCOM 7-35
 SETCURSOR 7-273
 SETDISK 7-277
 SETHEADING
 (SETH) 7-278

SETPAL 7-280
 SETPC 7-281
 SETPEN 7-197, 7-282
 SETPOS 7-284
 SETPRECISION 5-13,
 7-286
 SETREAD 7-287
 SETREADPOS 7-289
 .SETSCREEN 7-36
 .SETSCRUNCH 7-37
 SETSHAPE 7-291
 SETTC 7-292
 SETTEXT 7-176, 7-295
 SETUP 6-7, 6-10, 6-12
 SETWIDTH 7-296, 7-338
 SETWRITE 4-15, 7-298
 SETWRITEPOS 4-18,
 7-300
 SETX 7-301
 SETY 7-303
 SHAPE 7-304
 shift key 1-7, 7-11
 Shift-PrtSc 4-21, 7-267
 SHORTQUIZ 7-317
 SHOW 7-305
 SHOWARGS B-9
 SHOWLINES B-9
 SHOWNP 7-306
 SHOWTURTLE (ST) 7-307
 significant digits 7-227
 SIN 7-308
 SINE 7-352
 single stepping B-8
 SIREN 7-332
 SLITHER 7-61
 SNAKE 7-61
 SNAP 7-310
 sound (see TONE)
 source diskette 1-15
 space bar 1-10
 special
 characters 2-16
 keys 7-9
 words 7-12
 SPI 7-65, 7-257
 SPORTS 7-167
 SQ 7-312
 SQRT 7-312
 SQUARE 7-133, 7-262,
 7-328
 SQUARE.WITH
 TAIL 7-319
 SQUIRAL 7-136, 7-177
 STAMP 7-313
 STARTIMER 7-32
 starting Logo 1-11, 1-20, 7-30
 STARTUP file E-2
 STARTUP variable E-4
 STEER 7-151
 STEP B-8
 STEPPER B-8
 STOP 7-315
 Stopbits 7-35
 STORE 7-70, 7-299
 STORY 6-5, 6-7, 6-10, 6-12
 SUBMOUNTAIN 7-188
 subprocedure 2-4
 subroutine 7-26, C-8
 subtraction 7-15, 7-89
 SUFFIX 7-340
 SUM 7-316
 superprocedure 2-4
 SURPRISE 7-50
 .SYSTEM 7-12, 7-52, 7-207
 system reset 1-8

T

TAB 7-82
TAKEOFF 7-152
TALK 7-104
TAN 7-79
target diskette 1-16
TEACH B-10
TEST 7-317
TEXT 7-83, 7-318
text field 7-176, 7-293
text screen
 erasing 7-66, 7-67
 saving 4-11, 4-20, 7-267
TEXTCOLOR (TC) 7-320
TEXTSCREEN (TS) 7-322
THING 7-171, 7-324
THROW 7-60
time
 procedure using 7-32
 ticks 7-331, 7-337
TO 2-3, 7-327
TONE 7-329
top level 1-13, 2-4
TOPELVEL 7-60,
 7-195, 7-326
TOWARDS 5-20, 7-333
TRAIL 7-313, B-13
translating angles 7-203
TRIANGLE 7-55
trigonometric functions 7-79
TRUE 7-144
TRY 7-117
TRYAGAIN B-11
TURN 7-151

turtle

 degrees 5-20
 field 7-176
 shape 7-291, 7-304,
 7-310, 7-313
TURTLE 7-288, 7-291
TYPE 7-334

U

UNBURY 7-336
UNSTEP B-9
updating a file 4-18

V

VALPKG 7-205, 7-223
variables
 assigning values 2-11, 2-12,
 7-171, 7-324
 global 2-12
 local 2-12
VEE 7-217
VOWELP 7-175

W

WAIT 7-337
WALK 7-196
WEATHER 7-172
WELCOME 2-4
WELCOME.FOREVER 2-4
WHICH 7-189, B-14

WHILE 7-261, B-14
WHITE B-7
WIDTH 7-338
WINDOW 7-339
WIPEOUT B-11
word 7-8
WORD 7-340
WORD? 7-183
WORDP 7-184, 7-342
words 2-7, 2-16
workspace 4-5, 7-98, 7-265
WRAP 7-344
WRITEDATA 7-124
WRITEINFO 4-15
WRITEOFP (WRITE END
Of File) 7-345
WRITEPOS (WRITE
POSition) 7-346
WRITER 7-348

X

x-coordinate 7-93, 7-349
XCOR 7-349

Y

y-coordinate 7-93, 7-351
YCOR 7-351
YESNO 7-167

4

4 PEL B-6



Reader's Comment Form

Logo

1502222

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

Tape

Please do not staple

Tape

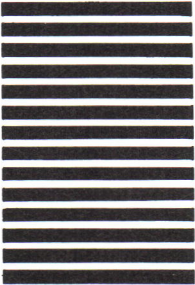
Fold here

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-W
BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES





Reader's Comment Form

Logo

1502222

Your comments assist us in improving the usefulness of our publication; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

Tape

Please do not staple

Tape

Fold here

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-W
BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 321 BOCA RATON, FLORIDA 33432

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Continued from inside front cover

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) or cassette(s) not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette(s) or cassette(s) which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL

DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328-W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.



International Business Machines Corporation

P.O. Box 1328-W
Boca Raton, Florida 33432

1502222

Printed in United States of America